УДК 373.167.1:002
ББК 32.81я72
К96

Кушниренко А. Г.
К96 Информатика. 7—9 кл.: Учеб. для общеобразоват. учеб. заведений / А. Г. Кушниренко, Г. В. Лебедев, Я. Н. Зайдельман. — 3-е изд., стереотип. — М.: Дрофа, 2002. — 336 с.: ил.

ISBN 5—7107—5283—5

Новый учебник информатики продолжает линию школьного курса информатики, начатую в 1985 году академиком А. П. Ершовым. Учебник во многом опирается на предыдущий учебник этого коллектива (*Кушниренко А. Г. и др.* Основы информатики и вычислительной техники. М.: Просвещение, 1990—1996) и развивает его основные идеи, адаптированные для изучения в 7—9 классах. Основное внимание авторы уделяют общим принципам обработки информации, независимым от конкретной технической базы. Специализированное программное обеспечение (система Кумир) работает на всех типах компьютеров, имеющихся в российских школах.

Учебник включен в Федеральный перечень.

УДК 373.167.1:002
ББК 32.81я72

Учебное издание

Кушниренко Анатолий Георгиевич
Лебедев Геннадий Викторович
Зайдельман Яков Наумович

ИНФОРМАТИКА
7—9 классы

Учебник для общеобразовательных учебных заведений

Зав. редакцией *М. Г. Циновская*. Редактор *Г. А. Гухман*
Оформление *Л. П. Копачева*. Художник *В. А. Иващук*
Макет *М. Г. Мицкевич*. Художественный редактор *М. Г. Мицкевич*
Технический редактор *Н. И. Герасимова*
Компьютерная верстка *С. Л. Мамедова*. Корректор *Е. Е. Никулина*

Изд. лиц. № 061622 от 07.10.97.
Подписано к печати 26.08.02. Формат 60x90$^1/_{16}$. Бумага типографская. Гарнитура «Таймс». Печать офсетная. Усл. печ. л. 21,0. Тираж 20 000 экз. Заказ 4210176.

ООО «Дрофа». 127018, Москва, Сущевский вал, 49.

По вопросам приобретения продукции издательства «Дрофа» обращаться по адресу: 127018, Москва, Сущевский вал, 49.
Тел.: (095) 795-05-50, 795-05-51. Факс: (095) 795-05-52.
Торговый дом «Школьник». 109172, Москва, ул. Малые Каменщики, д. 6, стр. 1А. Тел.: (095) 911-70-24, 912-15-16, 912-45-76.
Магазин «Переплетные птицы». 127018, Москва, ул. Октябрьская, д. 89, стр. 1. Тел.: (095) 912-45-76.

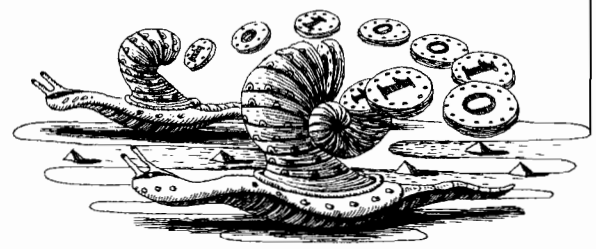
Отпечатано с готовых диапозитивов в ФГУИПП «Нижполиграф».
603006, Нижний Новгород, ул. Варварская, 32.

ISBN 5—7107—5283—5

© ООО «Дрофа», 2000

ГЛАВА 1

Информация и компьютеры



Мы начинаем знакомиться с новым предметом — информатикой. ***Информатика*** изучает методы представления, накопления, передачи и обработки информации с помощью ***компьютеров***. Что же такое информация, что такое компьютер и как компьютер обрабатывает информацию?

1 Информация

1.1. Информация — первичное, неопределяемое понятие информатики

Слово «информация» мы обычно понимаем как сообщение, сведения. Информацию помещают в газетах, передают по радио и телевидению. Но это — очень узкое понимание. Информация содержится в любом тексте, изображении, звуке и не только в них. Информация есть в рельефе ключа, в структуре сложной биологической молекулы, в радиосигналах, передаваемых на космический корабль. Информация в рельефе ключа позволяет открыть с его помощью определенный (свой) замок; информация в радиосигналах с Земли включает двигатель на космическом корабле и переводит корабль на другую орбиту; информа-

ция, запечатленная в структуре биологической молекулы, позволяет живой клетке производить определенные белки для новых тканей или для уничтожения попавших в организм микробов.

Так что же такое «информация»? Увы, этот термин в информатике является первичным, неопределяемым. Информатика изучает свойства информации, но строгого определения этого понятия не дает.

Такое положение не является чем-то необычным. Например, в геометрии не дается определений точки и прямой, они считаются первичными, неопределяемыми понятиями, но это не мешает изучать их свойства, решать задачи, доказывать теоремы.

1.2. Компьютер — универсальная машина для обработки информации

В современном обществе компьютеры применяются очень широко. Без них не обойтись в процессе подготовки к изданию книг и журналов, в научных и инженерных расчетах, при оформлении телепередач и во многих-многих других случаях.

Но все самые разнообразные применения компьютеров объединяет одно: что бы ни делал человек с помощью компьютера, это всегда будет работа с какой-то информацией. Ведь тексты, числа, изображения, звуки, все, что мы обрабатываем с помощью компьютера, — это различные формы представления информации.

Компьютер — ***универсальная машина для обработки информации***. Слово «универсальная» означает, что эта машина может обрабатывать информацию различного вида (тексты, изображения, числа, звуки и т. д.) и применяться в самых разных видах человеческой деятельности.

1.3. Как компьютер хранит информацию

Одну и ту же информацию можно представить по-разному. В старинном телеграфе, например, информация кодировалась и передавалась с помощью азбуки морзе — в виде последовательностей из точек и тире. В книгах и газетах информация представлена в виде текстов и изображений. Фразу, закодированную точками и тире азбуки морзе, можно напечатать буквами на пишущей машинке, произнести, передать жестами и т. д.

Память компьютера удобно упрощенно представить в виде очень большого количества маленьких переключателей, каждый из которых находится в одном из двух состояний. Физически эти состояния могут быть реализованы различными способами: намагничено или не намагничено, включено или выключено, высокое или низкое напряжение и т. д. Принято обозначать одно состояние цифрой 0, а другое — цифрой 1. Каждый такой переключатель называется ***бит*** (от англ. *bit — BInary digiT* — двоичная цифра). Цифры 0 и 1 тоже часто называют битами.

Оказывается, с помощью последовательности битов можно представить самую разнообразную информацию. Это представление называется ***двоичным кодированием***.

1.4. Двоичное кодирование чисел

Предположим, что у нас есть 4 бита. Меняя их состояния, мы можем получить 16 различных комбинаций, которые легко перечислить и перенумеровать (табл. 1).

ТАБЛИЦА 1. **Комбинации из 4 бит и соответствующие им числа**

0	1	2	3	4	5	6	7
0000	0001	0010	0011	0100	0101	0110	0111
8	9	10	11	12	13	14	15
1000	1001	1010	1011	1100	1101	1110	1111

Можно договориться, что данные комбинации бит соответствуют указанным числам. Существуют специальные правила (вы познакомитесь с ними несколько позже), которые дают возможность определить двоичный код любого натурального числа. При этом N бит позволяют закодировать 2^N различных чисел. Например, используя 16 бит, можно закодировать числа от 0 до 65 535, этого оказывается достаточно для многих расчетов.

Существуют также специальные способы для кодирования отрицательных и дробных чисел.

1.5. Двоичное кодирование текстов

При двоичном кодировании текстовой информации каждому символу сопоставляется его ***код*** — последовательность из фиксированного количества нулей и единиц. В большинстве современных ЭВМ каждому символу соответствует последовательность из 8 нулей и единиц, называемая ***байтом*** (англ. *byte*). Всего существует $2^8 = 256$ разных последовательностей из 8 нулей и единиц — это позволяет закодировать 256 разных символов, например большие и малые буквы русского и латинского алфавитов, цифры, знаки препинания и т. д. Соответствие байт и символов задается с помощью таблицы, в которой для каждого кода указывается отвечающий ему символ. Такая таблица называется ***кодовой таблицей*** или просто ***кодировкой***.

Обычно при записи кодовой таблицы в ней приводят не только двоичные коды символов, но и соответствующие им десятичные числа. Когда говорят, например, что код символа «!» равен 33, то имеется в виду, что двоичный код этого символа 00100001 соответствует десятичному числу 33.

Наиболее распространена сегодня кодировка ASCII (*American Standard Code for Information Interchange* — американский стандартный код для обмена информацией). Эта кодировка включает в себя всего 128 символов: латинские буквы, цифры, знаки препинания, некоторые служебные символы. Поскольку именно эти символы чаще всего используются при международном обмене информацией, кодировка ASCII стала фактическим мировым стандартом.

Фрагмент кодировки ASCII приведен в таблице 2.

Коду 00100000 в этой таблице соответствует ***пробел*** — пустой промежуток величиной в один символ, который используется для отделения одного слова от другого.

На основе ASCII в разных странах создают ***национальные кодировки***, заполняя оставшиеся 128 символов буквами национальных алфавитов и некоторыми дополнительными символами.

Для русского языка существует несколько различных кодировок. Все они совпадают с ASCII при кодировании латинских букв, цифр и основных знаков препинания, но отличаются кодами русских букв.

ТАБЛИЦА 2. **Фрагмент кодировки ASCII**

Двоичный код	Десятичный код	Символ	Двоичный код	Десятичный код	Символ
00100000	32	пробел	01000000	64	@
00100001	33	!	01000001	65	A
00100010	34	"	01000010	66	B
00100011	35	#	01000011	67	C
00100100	36	$	01000100	68	D
00100101	37	%	01000101	69	E
00100110	38	&	01000110	70	F
00100111	39	‘	01000111	71	G
00101000	40	(	01001000	72	H
00101001	41	)	01001001	73	I
00101010	42	*	01001010	74	J
00101011	43	+	01001011	75	K
00101100	44	,	01001100	76	L
00101101	45	-	01001101	77	M
00101110	46	.	01001110	78	N
00101111	47	/	01001111	79	O
00110000	48	0	01010000	80	P
00110001	49	1	01010001	81	Q
00110010	50	2	01010010	82	R
00110011	51	3	01010011	83	S
00110100	52	4	01010100	84	T
00110101	53	5	01010101	85	U
00110110	54	6	01010110	86	V
00110111	55	7	01010111	87	W
00111000	56	8	01011000	88	X
00111001	57	9	01011001	89	Y
00111010	58	:	01011010	90	Z
00111011	59	;	01011011	91	[
00111100	60	<	01011100	92	\
00111101	61	=	01011101	93	]
00111110	62	>	01011110	94	^
00111111	63	?	01011111	95	_

Наиболее распространены кодировки, которые называются КОИ-8 (КОИ сокращение от слов ***Код Обмена Информацией***) и Windows-1251 (по названию компьютерной системы Windows, где используется эта кодировка).

В таблицах 3 и 4 можно посмотреть, как кодируются русские буквы в этих кодировках.

Например, в кодировке КОИ-8 большая русская буква «М» имеет код 11101101, буква «И» — код 11101001, буква «Р» — код 11110010, буква «У» — код 11110101. Таким образом, слово «МИР» кодируется последовательностью из 24 бит

111011011110100111110010

или из трех чисел

237 233 242,

а фраза «МИРУ МИР» — последовательностью

111011011110100111110010111101010010000011101101111010011111 0010

или

237 233 242 245 32 237 233 242

Конечно, при практической работе с компьютером совершенно не нужно помнить эти таблицы, все преобразования совершаются автоматически.

1.6. Двоичное кодирование изображений

Последовательностями нулей и единиц можно закодировать и графическую информацию. Всмотритесь в газетную фотографию и вы увидите, что она состоит из мельчайших точек. У разного полиграфического оборудования густота этих точек разная — фотографии в одних газетах намного четче, чем в других. В большинстве старых газет фотографии содержат 24 точки на сантиметр длины, т. е. фотография размером 10×10 сантиметров состоит примерно из 60 тыс. точек. Если это только черные и белые точки, то каждую из них можно закодировать одним битом, а всю фотографию — последовательностью из 60 тыс. бит. Для повышения качества кодировки черно-белых изображений можно кодировать точки не одним битом, а несколькими. Два бита позволяют закодировать 4 оттенка точек: 00 — белый цвет, 01 — светло-серый, 10 — темно-серый, 11 — черный. Три бита позволяют закодировать 8 оттенков и т. д.

ТАБЛИЦА 3. **Русские буквы в кодировке КОИ-8**

Двоичный код	Десятичный код	Символ	Двоичный код	Десятичный код	Символ
11000000	192	ю	11100000	224	Ю
11000001	193	а	11100001	225	А
11000010	194	б	11100010	226	Б
11000011	195	ц	11100011	227	Ц
11000100	196	д	11100100	228	Д
11000101	197	е	11100101	229	Е
11000110	198	ф	11100110	230	Ф
11000111	199	г	11100111	231	Г
11001000	200	х	11101000	232	Х
11001001	201	и	11101001	233	И
11001010	202	й	11101010	234	Й
11001011	203	к	11101011	235	К
11001100	204	л	11101100	236	Л
11001101	205	м	11101101	237	М
11001110	206	н	11101110	238	Н
11001111	207	о	11101111	239	О
11010000	208	п	11110000	240	П
11010001	209	я	11110001	241	Я
11010010	210	р	11110010	242	Р
11010011	211	с	11110011	243	С
11010100	212	т	11110100	244	Т
11010101	213	у	11110101	245	У
11010110	214	ж	11110110	246	Ж
11010111	215	в	11110111	247	В
11011000	216	ь	11111000	248	Ь
11011001	217	ы	11111001	249	Ы
11011010	218	з	11111010	250	З
11011011	219	ш	11111011	251	Ш
11011100	220	э	11111100	252	Э
11011101	221	щ	11111101	253	Щ
11011110	222	ч	11111110	254	Ч
11011111	223	ъ	11111111	255	Ъ

ТАБЛИЦА 4. **Русские буквы в кодировке Windows-1251**

Двоичный код	Десятичный код	Символ	Двоичный код	Десятичный код	Символ
11000000	192	А	11100000	224	а
11000001	193	Б	11100001	225	б
11000010	194	В	11100010	226	в
11000011	195	Г	11100011	227	г
11000100	196	Д	11100100	228	д
11000101	197	Е	11100101	229	е
11000110	198	Ж	11100110	230	ж
11000111	199	З	11100111	231	з
11001000	200	И	11101000	232	и
11001001	201	Й	11101001	233	й
11001010	202	К	11101010	234	к
11001011	203	Л	11101011	235	л
11001100	204	М	11101100	236	м
11001101	205	Н	11101101	237	н
11001110	206	О	11101110	238	о
11001111	207	П	11101111	239	п
11010000	208	Р	11110000	240	р
11010001	209	С	11110001	241	с
11010010	210	Т	11110010	242	т
11010011	211	У	11110011	243	у
11010100	212	Ф	11110100	244	ф
11010101	213	Х	11110101	245	х
11010110	214	Ц	11110110	246	ц
11010111	215	Ч	11110111	247	ч
11011000	216	Ш	11111000	248	ш
11011001	217	Щ	11111001	249	щ
11011010	218	Ъ	11111010	250	ъ
11011011	219	Ы	11111011	251	ы
11011100	220	Ь	11111100	252	ь
11011101	221	Э	11111101	253	э
11011110	222	Ю	11111110	254	ю
11011111	223	Я	11111111	255	я

Если изображение цветное, то каждая комбинация бит означает какой-то цвет или оттенок. В современных компьютерах при получении изображений высокого качества цвет одной точки кодируют с помощью 24 бит. Это позволяет получить $2^{24} = 16\,777\,216$ (более 16 миллионов!) цветов. При таком большом количестве цветов и маленьком размере каждой отдельной точки человеческий глаз воспринимает изображение как единое целое, не замечая, что оно состоит из отдельных точек.

1.7. Единицы измерения информации

Для измерения длины, массы, времени, силы тока и т. д. придуманы приборы и процедуры измерения. Чтобы узнать длину стержня, достаточно приложить к нему линейку с делениями, время можно измерить с помощью часов.

А как узнать количество информации в сообщении, в каких единицах эту информацию измерять? Для двоичных сообщений в качестве такой числовой меры используется количество бит в сообщении. Это количество называют ***информационным объемом*** сообщения. Например, сообщение «МИРУ МИР» в коде КОИ-8 имеет информационный объем 8 байт (64 бит).

Биты и байты используются также для измерения «емкости» памяти и скорости передачи двоичных сообщений. Скорость передачи измеряется количеством бит, передаваемых в секунду (например, 19 200 бит/с).

Наряду с битами и байтами для измерения количества информации в двоичных сообщениях используются и более крупные единицы:

1 Кбит (один килобит) = 2^{10} = 1024 бит (≈ 1 тыс. бит);
1 Мбит (один мегабит) = 20^{20} = 1 048 576 бит (≈ 1 млн бит);
1 Гбит (один гигабит) = $2^{30} \approx 10^9$ бит (≈ 1 млрд бит);

1 Кбайт (один килобайт) = 2^{10} = 1024 байт (≈ 1 тыс. байт);
1 Мбайт (один мегабайт) = 2^{20} = 1 048 576 байт (≈ 1 млн байт);
1 Гбайт (один гигабайт) = 2^{30} (≈ 1 млрд байт).

К единицам измерения многих физических величин мы привыкли, и нам не нужно пояснять, что такое 1 миллиметр или

10 километров. А бит, байт, килобайт, мегабайт, гигабайт — много это или мало?

В коде ASCII каждая буква, знак препинания, пробел — это 1 байт. На странице нашего учебника помещается чуть меньше 40 строк, в каждой строке — примерно 60 знаков (60 байт). Таким образом, полностью заполненная текстом страница учебника имеет информационный объем около 2400 байт (2,4 Кбайт). Средняя страница содержит около 2 Кбайт, а весь учебник — чуть больше 0,6 Мбайт текста. В Большой Советской Энциклопедии примерно 120 Мбайт. В одном номере четырехстраничной газеты — 150 Кбайт, а если собрать по одному номеру всех газет, выходящих в нашей стране, то в них будет уже несколько гигабайт информации. Если человек говорит по 8 часов в день без перерыва, то за 70 лет жизни он наговорит около 10 Гбайт информации (это 5 млн страниц — стопка листов бумаги высотой 500 м).

В одном черно-белом телевизионном кадре (при 32 градациях яркости каждой точки) заключено примерно 300 Кбайт информации. Цветной кадр, образованный из трех кадров основных цветов (красный, синий, зеленый), содержит уже около мегабайта информации, а 1,5-часовой цветной телевизионный художественный фильм (при частоте 25 кадров в секунду) — 135 Гбайт.

Если на условной шкале изобразить бит примерно одним миллиметром (точнее, отрезком длиной 1,25 мм), то байт в этом масштабе будет представлен сантиметром, килобайт — десятиметровым отрезком, мегабайт — десятикилометровым, ну а гигабайт вытянется в 10 000 километров — это расстояние от Москвы до Владивостока. Как видите, диапазон, который охватывают единицы измерения информации, очень велик.

1.8. Термин «информация» в информатике

Обычно слово «информация» ассоциируется у нас со смыслом, значимостью сообщения. С этой точки зрения телеграмма о дате приезда не несет никакой информации, если мы эту дату уже знаем. Смысл и значимость, однако, понятия человеческие, субъективные. Информатика же изучает обработку информации с помощью компьютера, т. е. с научно-технической точки зрения. Поэтому трактовка этого понятия в информатике отличается от обыденной.

Смысл информации, ее ценность, значимость — все это важно для человека. Для компьютера информация — это всего лишь набор бит. Соответственно информационный объем двоичного сообщения — это чисто техническая характеристика. Он показывает, сколько ресурсов потребуется компьютеру для хранения и передачи данного сообщения, но никак не отражает его важность. Для хранения и передачи бессмысленного или ошибочного сообщения компьютер потратит столько же усилий, сколько он потратит на работу с важным осмысленным сообщением того же объема. Информационный объем сообщения можно сравнить с весом перевозимого груза: чем больше вес груза, тем дороже его перевозка, но ценность и важность груза не определяются его весом.

Когда компьютер обрабатывает информацию, он совершает некоторые простые преобразования, которые могут быть описаны ***строгими формальными правилами***. Вот некоторые примеры таких преобразований: замена одной буквы на другую в тексте; замена нулей на единицы, а единиц на нули в последовательности бит; сложение двух чисел, когда из информации, представляющей слагаемые, получается результат — сумма.

Слова «обработка информации», таким образом, вовсе не подразумевают восприятие информации или ее осмысление. Компьютер — всего лишь машина, способная исключительно к технической, машинной обработке информации.

Только человек в состоянии наделять информацию смыслом. Человек придумывает способы кодирования информации, которые сопоставляют последовательности бит какие-то осмысленные значения. Он изобретает методы обработки информации, которые преобразуют эти последовательности бит так, чтобы в результате получилась нужная для человека информация.

А компьютер просто выполняет преобразования, заданные человеком, но делает это быстро и безошибочно. В нашем курсе мы будем учиться давать компьютеру задания на обработку информации, чтобы получать осмысленный для нас результат.

Задача 1. Сколько двоичных цифр (бит) необходимо, чтобы закодировать одну школьную оценку?

■ ***Решение.*** Всего существует 5 школьных оценок. Два бита позволяют закодировать $2^2 = 4$ комбинации. $4 < 5$, значит, двух бит будет недостаточно. Три бита позволяют закодировать $2^3 = 8$ комбинаций. $8 > 5$, следовательно, трех бит хватит.

Коротко это можно записать так: $2^2 < 5 < 2^3$.

■ ***Ответ.*** 3 бита.

Задача 2. Петя и Коля играют в «черный ящик». Петя задумывает правило преобразования текстовой информации. Коля может задавать Пете любые тексты и узнавать результаты преобразования. Задача Коли — отгадать задуманное правило. Перед вами протокол игры: вопросы Коли и ответы Пети (табл. 5). Отгадайте правило.

ТАБЛИЦА 5. **Протокол игры в «черный ящик»**

№	Вопрос	Ответ
1	сова	4
2	корова	6
3	банк	2
4	собака	4
5	яблоко	0

■ ***Решение.*** Надо подобрать такое правило, которое подходило бы сразу для всех примеров протокола.

По первым двум вопросам можно предположить, что речь идет о количестве букв в слове, но на третьем вопросе выясняется, что это не так.

Еще одно предположение — удвоенное количество слогов. Эта гипотеза хорошо объясняет первые три вопроса, но затем выясняется, что она тоже неверна.

А что, если число указывает не на количество, а на номер буквы? Действительно, в первых четырех случаях на этом месте оказывается буква «а». В слове «яблоко» буквы «а» нет, поэтому в ответе на это слово появился нуль.

Наша гипотеза объясняет все ответы Пети, значит, она может быть верной.

■ ***Ответ.*** Петя определяет номер первой буквы «а» в слове.

ЗАДАЧИ И УПРАЖНЕНИЯ

1. Приведите несколько бытовых примеров получения, хранения, передачи, обработки, использования информации.

2. а) Единица — очень редкая в школе оценка, по смыслу почти неотличимая от двойки. Сколько бит потребуется для кодирования оценок, если запретить ставить школьникам единицы?
б) В некоторых школах учителя после оценки иногда пишут «плюс» или «минус» (например: 5+, 3– и т. п.). Сколько бит потребуется для кодирования одной отметки при такой системе?

3. Человек способен различить примерно 100 градаций яркости. Сколько бит необходимо, чтобы закодировать конкретное значение яркости?

4. Колода для игры в преферанс состоит из 32 карт: 4 масти (пики, трефы, бубны, черви) по 8 карт в каждой (7, 8, 9, 10, валет, дама, король, туз). Сколько бит необходимо, чтобы закодировать одну карту? Придумайте способ кодирования, который позволит по заданной карте определить ее двоичный код, а по коду — карту.

5. Закодируйте в коде ASCII: а) текст «VEDI, VIDI, VICI»; б) текст «TO BE OR NOT TO BE»; в) запись числа «1990»; г) выражение (последовательность символов) «2X + Y = 0».
У к а з а н и е: не забудьте закодировать пробелы!

6. Закодируйте в коде КОИ-8 свои имя и фамилию. Запишите результат в виде
а) последовательности битов;
б) десятичных значений байтов.

7. Какие последовательности символов закодированы следующими кодами в КОИ-8:
а) 111011011110100111101101;
б) 001100010011011000110001;
в) 010000010101100000101011010000100011110100110000;
г) 0100001101001111010011010101000001010101010101000100010101010010?

8. Определите, в какой кодировке записан текст, и расшифруйте его:
а) 232 237 244 238 240 236 224 242 232 234 224;
б) 196 226 238 232 247 237 251 233 32 234 238 228;
в) 203 210 207 203 207 196 201 204 32 231 197 206 193.

9. Петя и Коля пишут друг другу электронные письма в кодировке КОИ-8. Однажды Петя ошибся и отправил письмо в кодировке Windows. Коля получил письмо и как всегда прочитал его в КОИ-8. Получился бессмысленный текст, в котором часто повторялось слово БНОПНЯ. Какое слово было в исходном тексте письма?

10. Считая, что каждый символ кодируется одним байтом, определите приблизительно информационный объем:
а) вашего школьного дневника;
б) учебников в вашем портфеле;
в) классного журнала.

11. Придумайте эксперимент, позволяющий узнать, сколько символов в секунду вы читаете, пишете, произносите. Проведите этот эксперимент.

12. Подсчитайте приблизительно, сколько байт вы прочли, написали, произнесли за время учебы в школе.

13. Придумайте способ передать изображение по телефону. Сколько времени понадобится, чтобы передать фотографию размером 4×6 см?

14. В приведенных ниже последовательностях каждый следующий элемент получен из предыдущих по некоторому строгому правилу. Угадайте это правило:
а) а, б, в, г, д, е, ...;
б) 1, 2, 3, 4, 5, 6, 7, ...;
в) 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, ...;
г) 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...;
д) 1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 0, 1, 1, 1, 2, 1, ...;
е) победа, обеда, беда, еда, ...;
ж) о, д, т, ч, п, ш, с, в, д, д, ...;
з) 1, 11, 21, 1211, 111221, 312211, 13112221, ...

15. Для чего в ответ примера 2 включено слово «первой»? Какой пример в протоколе доказывает правильность этого предположения?

16. Отгадайте правило «черного ящика» по заданным протоколам:
а) А → Б; мама → нбнб; ЭВМ → ЮГН; язык → аиъл;
б) А → 1; мама → 4; ЭВМ → 3; язык → 4;
в) А → 0 + 1; мама → 2 + 2; ЭВМ → 2 + 1; язык → 2 + 2;
г) А → А; мама → амам; ЭВМ → ВМЭ; язык → зыкя.

17. При игре в «черный ящик» можно задумывать правило обработки чисел, а не текстов. Отгадайте несколько таких правил по заданным протоколам:
а) 1 → 0; 5 → 4; 0 → –1; 1990 → 1989;
б) 1 → 0; 2 → 0; 10 → 9; 3 → 3; 20 → 18; 1990 → 1989;
в) 1 → 1; 7 → 1; 10 → 2; 187 → 3; 1990 → 4;
г) 1 → 0; 8 → 2; 16 → 1; 1990 → 3; 1989 → 4; 100 → 2; 108 → 3; 6 → 1; 7 → 0; 23 → 0; 50 → 1.

18. Опишите правило преобразования текста, которое фразы вида «Тебя зовут Вася?» превращает во фразы вида «Это тебя зовут Вася!». Примените это преобразование к нескольким другим фразам.

19. Даны: лист миллиметровки, иголка, лупа с десятикратным увеличением. Придумайте способ записать на листе миллиметровки как можно больше информации.

20. Даны: лист тонкого картона и тупое шило. Придумайте способ записать на листе информацию так, чтобы ее можно было считать в темноте, на ощупь.

21. Коля сидит дома, делает уроки и не подходит к телефону. Петя набирает Колин номер, ждет некоторое время и кладет трубку. Затем снова набирает Колин номер и т. д. Придумайте способ передачи информации с помощью таких звонков.

2 Персональный компьютер

2.1. Краткая история вычислительной техники

С древнейших времен человек конструирует себе в помощь различные приспособления для облегчения вычислений. Еще в V в. до н. э. греки и египтяне использовали абак — устройство, похожее на русские счеты.

В 40-е гг. XVII в. один из крупнейших ученых в истории человечества — математик, физик, философ и богослов Блез Паскаль изобрел и изготовил механическое устройство, с помощью которого можно складывать числа в десятичной системе счисления (рис. 1).

Механическое устройство, позволяющее не только складывать числа, но и умножать их, было изобретено другим великим математиком и философом — Готфридом Вильгельмом Лейбницем в конце XVII в. (см. рис. 1). В работах Лейбница шла речь и о механическом устройстве, способном оперировать со словами и выражаемыми ими понятиями.

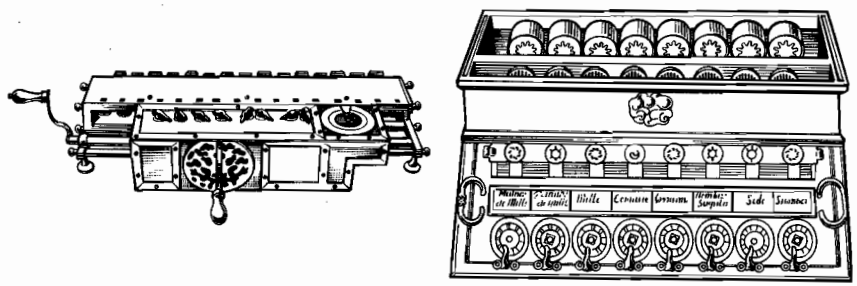


Рис. 1. Арифмометры Паскаля и Лейбница

Наряду с устройствами, предназначенными для вычислений, развивались и механизмы для ***автоматической работы по заданной программе*** (музыкальные автоматы, шарманки, часы с боем и т. п.). В шарманку, например, помещали диски с расположенными по-разному штырьками — в зависимости от расположения штырьков звучала та или иная мелодия. В ткацком станке Жаккарда узор ткани задавался с помощью дырочек в тонких картонных картах (перфокартах). Для смены узора достаточно было по-другому пробить дырочки в перфокарте.

Важный прогресс в области вычислительной техники связан с именем Чарльза Беббиджа (середина XIX в.). Соединив идею механической арифметической машины Лейбница с идеей программного управления, Беббидж разработал проект машины, названной им «аналитической». Этот проект не был реализован, однако по своим возможностям машина Беббиджа не уступала первым ЭВМ: в ней была предусмотрена память для хранения 1000 чисел по 50 десятичных знаков; арифметические операции выполнялись в соответствии с программой, записанной на жаккардовых перфокартах. В программе можно было задать автоматическое повторение группы арифметических операций, а также выполнение группы операций только при определенном условии. С машиной Беббиджа связано появление профессии программиста. Первым программистом мира стала дочь поэта Дж. Байрона Ада Лавлейс. Программы, написанные Адой Лавлейс, предназначались для вычисления значений некоторых числовых функций.

В конце 30-х гг. XX в. американцы Дж. Атанасов (болгарин по происхождению) и К. Берри построили ЭВМ, включавшую в себя электронную память и электронное устройство сложения и вычитания, а также ряд механических компонентов. Эта машина еще не была универсальной, однако область ее применения была значительно больше, чем у механических арифмометров. В 1942 г. была построена усовершенствованная модель ЭВМ Атанасова—Берри, предназначенная прежде всего для решения систем линейных уравнений (до 30 уравнений с 30 неизвестными).

До конца 40-х гг. был создан ряд других машин, более мощных и совершенных. В 1946 г. в научной статье трех американских авторов — Дж. фон Неймана, Г. Голдстайна и А. Бернса — были изложены основные принципы построения универсальной ЭВМ, использующей одну и ту же память для хранения и обрабатываемых данных, и программы вычислений. Первая машина,

реализующая эти принципы, — ЭВМ EDSAC — была построена в 1949 г. под руководством М. В. Уилкса в Англии, в Кембриджском университете (в нем когда-то учился Беббидж). Через год была построена универсальная ЭВМ EDVAC в США.

Основоположником отечественной вычислительной техники был Сергей Алексеевич Лебедев. Под его руководством создавались первые отечественные ЭВМ: в 1951 г. в Киеве — МЭСМ (малая электронная счетная машина) и в 1952 г. в Москве — БЭСМ (быстродействующая электронная счетная машина). Почти одновременно с БЭСМ были разработаны ЭВМ М-2 и «Стрела».

С середины 50-х гг. началось бурное развитие вычислительной техники. Появилось множество моделей компьютеров, их стали производить и использовать во всех развитых странах.

В начале 80-х гг. появились первые персональные компьютеры — небольшие машины, которые можно разместить на рабочем столе. Поначалу их не принимали всерьез, многим специалистам казалось, что это игрушка, у которой нет будущего. Но успехи технологии привели к тому, что сегодняшние персональные компьютеры по своим возможностям заметно превосходят большие машины 10-летней давности.

В 90-е гг. персональные компьютеры стали преобладать в вычислительной технике. Сейчас в мире работают сотни миллионов персональных компьютеров, без них невозможно представить современное предприятие. Компьютер вошел и в быт, став для многих привычным и хорошо знакомым прибором.

Современный персональный компьютер способен делать сотни миллионов операций в секунду, т. е. десятки тысяч миллиардов операций в сутки. Однако встречаются и такие задачи, для решения которых нужно проделать десятки тысяч триллионов (1 триллион = 1000 миллиардов) операций. На одном персональном компьютере такая задача будет решаться несколько лет, и к моменту получения результата расчета он может оказаться никому не нужным. Например, прогноз погоды на следующую неделю нужно успеть «подсчитать» на ЭВМ за несколько часов. А подсчет, занимающий неделю или более, не имеет никакого практического значения.

До недавнего времени сложные расчеты проводились на так называемых суперЭВМ — очень дорогих специализированных компьютерах, выпускаемых в единичных экземплярах. К концу XX в. классические суперЭВМ — «динозавры компьютерного мира» — стали вымирать. Инженеры научились делать супер-

ЭВМ нового типа, соединяя для совместной работы сотни и тысячи персональных ЭВМ. Персональные ЭВМ производятся огромными тиражами и потому дешевы. Как следствие, суперЭВМ нового типа оказываются в десятки раз дешевле специализированных суперЭВМ той же вычислительной мощности.

2.2. Состав персонального компьютера

Любой компьютер обязательно включает в себя несколько основных частей: ***процессор, память*** и ***внешние устройства***.

Упрощенно работу персонального компьютера можно описать так: процессор выполняет программу, размещенную в памяти, оперируя с данными, находящимися в памяти. При этом процессор тоже управляет внешними устройствами и обменивается информацией с ними, а также с человеком и другими компьютерами.

Процессор — мозг машины. Он выполняет обработку информации и управляет работой всех остальных устройств компьютера (рис. 2).

Технически процессор современной персональной ЭВМ выполнен в виде ***микропроцессора*** — одной сверхбольшой интегральной схемы, состоящей из нескольких миллионов элементов. Микропроцессор — одна из самых сложных конструкций, созданных человеком.

Основная характеристика процессора — ***быстродействие*** (число выполняемых в секунду операций) и ***разрядность*** (количество бит, обрабатываемых процессором за одну операцию).

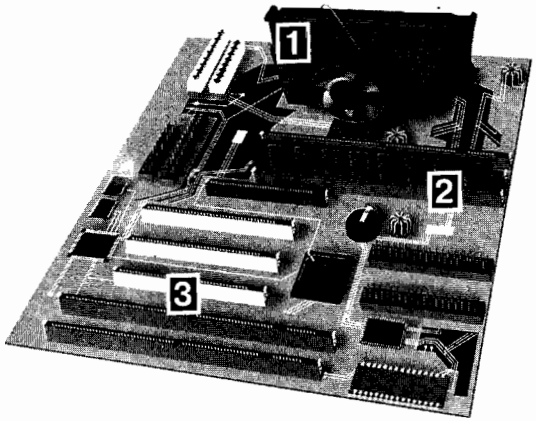


Рис. 2. Материнская плата компьютера:
1 — микропроцессор с радиатором и охлаждающим вентилятором; 2 — модуль памяти; 3 — разъемы для подключения внешних устройств.

Хотя процессор выполняет разные операции с разной скоростью, удобно говорить про некоторое усредненное быстродействие процессора. Первые компьютеры выполняли несколько тысяч операций в секунду. Лучшие машины 70-х гг. достигали быстродействия в несколько миллионов операций в секунду. Быстродействие среднего персонального компьютера начала XXI в. составляет несколько сот миллионов операций в секунду.

Память компьютера хранит информацию. Технически память компьютера (ее иногда называют ***оперативная память***) — это микросхемы небольшого размера.

Основная характеристика памяти — ее ***объем***. Чем больше объем оперативной памяти, тем больший объем информации процессор способен обрабатывать с максимально возможной скоростью. Память первых компьютеров составляла несколько сотен байт, первые персональные компьютеры имели несколько десятков килобайт памяти. Типичный объем памяти современного персонального компьютера — от 32 до 128 Мбайт.

Кроме оперативной памяти у компьютера может быть ***постоянная память*** (***ПЗУ***), содержимое которой устанавливается на заводе-изготовителе и в дальнейшем не изменяется.

2.3. Внешние устройства компьютера

Внешние устройства позволяют компьютеру обмениваться информацией с человеком и с другими компьютерами. Их разнообразие очень велико, и с развитием технологий все время появляются новые устройства. Мы рассмотрим только самые основные из них, наиболее часто встречающиеся.

Обычно компьютер обладает ***внешней памятью***, предназначенной для долговременного хранения информации (см. ниже). Самый широко используемый вид внешней памяти — ***магнитные диски***. В последнее время все большее распространение получают новые виды внешней памяти, в частности ***магнитооптические*** (***лазерные***) ***диски***, позволяющие хранить сотни мегабайт информации на сравнительно небольших носителях (рис. 3). Типичная внешняя память современного персонального компьютера составляет несколько гигабайт.

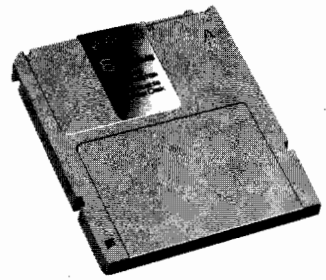


Рис. 3

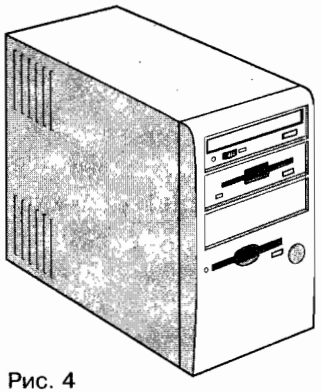


Рис. 4

В современных персональных компьютерах процессор, оперативная и внешняя память обычно находятся в общем корпусе, который называется ***системным блоком*** (рис. 4).

Основные устройства для взаимодействия компьютера с человеком — ***клавиатура, мышь*** и ***монитор***. Клавиатура и мышь предназначены для передачи информации от человека к компьютеру, монитор — от компьютера к человеку.

Для вывода информации на бумагу служит ***принтер***. С помощью ***сканера*** можно выполнить обратную операцию — ввести изображение с бумаги в компьютер.

Модем позволяет подключить компьютер к телефонной сети. Два компьютера, оборудованные модемами, могут обмениваться информацией по телефону.

Рис. 5

На телефонной станции может быть смонтирован мощный компьютер, соединенный с мировой сетью Интернет. Персональный компьютер, снабженный модемом, может соединяться с таким мощным компьютером по телефонной линии и оказаться включенным в Интернет на все время соединения.

В учреждениях все персональные компьютеры на рабочих местах сотрудников обычно связаны в единую локальную сеть. Для подключения персонального компьютера к сети учреждения предназначен ***адаптер локальной сети***.

Общая информационная схема персонального компьютера показана на рисунке 5.

2.4. Видеосистема персонального компьютера

Видеосистема состоит из видеоплаты и монитора. Видеоплата — это плата, на которой расположены видеопроцессор и видеопамять. Изображение полностью формируется в видеопамяти, а затем передается на экран монитора.

Изображение формируется из отдельных маленьких точек. Каждая такая точка называется ***пиксел***. Пикселы образуют прямоугольник, который называется ***растр***.

Растры бывают разного размера, например: 640×480, 800×600, 1024×768, 1152×864, 1280×1024 или 1600×1200 точек. Числа означают количество пикселов в строке и количество строк. Растр 1600×1200 — это 1600 точек в строке и 1200 таких строк. Размер растра называется ***разрешением***.

Для каждой точки растра указывается ее цвет. Цвет может задаваться одним байтом, в этом случае количество возможных цветов составляет 256 ($2^8 = 256$), двумя байтами — более 60 тыс. цветов ($2^{16} = 65\,536$) или тремя байтами — 16 млн цветов ($2^{24} = 16\,777\,216$).

От разрешения и количества цветов зависит качество изображения и объем необходимой видеопамяти. Подробнее об этом сказано в § 35.

2.5. Средства связи между компьютерами

В современном мире компьютер стал основным инструментом накопления и обработки информации, поэтому очень актуальной стала задача обмена информацией между компьютерами.

Для этого создаются ***сети***, объединяющие компьютеры и позволяющие им обмениваться информацией. Специальные компьютерные сети позволяют передавать информацию со скоростью до 100 Мбит в секунду.

Однако такие сети в нашей стране только начинают создаваться, они есть далеко не везде. Поэтому для связи удаленных (т. е. не находящихся рядом) компьютеров применяется телефонная сеть, которая сегодня есть повсюду.

Для подключения компьютера к телефонной сети необходимо специальное устройство — модем. Два компьютера, оснащенные модемами, могут «общаться» по телефону, передавая любую информацию.

Важнейшая характеристика модема — скорость передачи данных, которая измеряется в битах в секунду. Несколько лет назад чаще всего использовались модемы со скоростью 2400 бит/с, а скорость 9600 бит/с считалась очень высокой. Сегодня, как правило, используются модемы со скоростью 28 800 и 33 600 бит/с.

Рассмотрим пример решения задачи.

Задача. Видеоадаптер VGA, распространенный в начале 90-х гг., позволял получить 16-цветное изображение размером 640×480 точек. Сколько видеопамяти требуется для хранения такого изображения?

■ ***Решение.*** Для хранения каждой точки 16-цветного изображения необходимы 4 бита ($2^4 = 16$). Всего потребуется $640 \times 480 \times 4$ бит видеопамяти. Чтобы упростить вычисления, можно вести счет «по степеням двойки»:

$640 \times 480 \times 4 = 64 \times 10 \times 16 \times 3 \times 10 \times 4 = 2^6 \times 10 \times 2^4 \times 3 \times 10 \times \times 2^2 = 2^{12} \times 3 \times 100$ бит $= 2^9 \times 3 \times 100$ байт $= 2^{-1} \times 3 \times 100$ Кбайт $=$ $= 150$ Кбайт.

■ ***Ответ.*** 150 Кбайт.

ЗАДАЧИ И УПРАЖНЕНИЯ

1. а) Первый отечественный домашний компьютер БК-0010 имел оперативную память 16 Кбайт. Сколько страниц нашего учебника можно было разместить в этой памяти?

б) Компьютер, на котором авторы готовили этот учебник, имеет оперативную память 16 Мбайт. Сколько страниц учебника можно разместить в этой памяти?

2. Дискета объемом 1440 Кбайт весит 20 г. Сколько будет весить набор дискет, необходимый для полного копирования информации с жесткого диска объемом 2 Гбайта?

3. На одном компакт-диске размещается 600 Мбайт информации. Сколько времени потребуется, чтобы передать всю эту информацию по компьютерной сети со скоростью 100 бит/с? А если вести передачу через модем со скоростью 28 800 бит/с?

4. На компакт-диске объемом 600 Мбайт размещена библиотека рисунков. Средний размер каждого рисунка — 30 Кбайт. Сколько времени потребуется на просмотр всех рисунков, если на просмотр одного рисунка уходит 5 с?

5. Для подготовки к печати художественных альбомов необходимо иметь возможность обработки изображений с разрешением 1680 × 1200 при 16 млн цветов. Хватит ли для этого 4 Мбайт видеопамяти?

6. Какое максимальное количество цветов может обеспечить видеопамять 1 Мбайт при разрешении 800 × 600?

7. Матричный принтер Epson печатает 60 символов в секунду в режиме высокого качества печати и 120 символов в секунду в режиме черновика. Сколько времени потребуется для печати всего нашего учебника в каждом из этих режимов? За какое время можно выполнить ту же работу на лазерном принтере, печатающем со скоростью 6 страниц в минуту?

3 Обработка текстов на ЭВМ

3.1. Компьютер — лучший инструмент обработки текстов

Информация играет огромную роль в жизни нашей цивилизации. С тех пор как несколько тысячелетий назад человек изобрел письменность, записи стали важнейшим средством передачи и сохранения информации. И сегодня, несмотря на бурное развитие технологий, связанных с нетрадиционными способами хранения информации (например, видео), она в основном по-прежнему существует в виде текстов.

В течение многих веков люди писали тексты одним и тем же способом — с помощью специального инструмента помещали знаки на носитель. Менялись инструменты и носители, но суть

процесса оставалась прежней. Конечно, писать ручкой по бумаге много удобнее, чем вырубать топором записи на скале, но ручка, как и топор, не позволяет, не оставив следов, исправить уже написанное. Как говорится, «что написано пером, не вырубишь топором».

В XV в. Иоганн Гутенберг изобрел книгопечатание. Это сделало возможным быстрое и дешевое размножение книг — их стали печатать, а не переписывать от руки. Но процесс *написания* текста остался прежним.

Качественные изменения в написании текстов произошли после появления пишущей машинки. Это изобретение получило широкое распространение в конце XIX — начале XX в. Использование пишущей машинки позволило человеку писать быстрее (ведь нажимать на клавиши быстрее, чем вырисовывать буквы), получать более аккуратные тексты, решить проблему неразборчивого почерка. Однако исправить написанное на машинке ничуть не проще, чем при использовании ручки.

Сегодня основным инструментом подготовки текстов стал компьютер. На любом компьютере обязательно есть ***текстовые редакторы***. С их помощью можно не просто набирать текст, как на пишущей машинке, но и выполнять многочисленные операции по его обработке: вносить любые исправления, не перепечатывая весь текст, вставлять, удалять и менять местами отдельные слова и целые фрагменты, автоматически заменять одно слово на другое и находить орфографические ошибки. Таким образом, подготовка текстов с помощью ЭВМ открывает качественно новые возможности, которых не было ранее.

Применение ЭВМ не заканчивается на этом этапе. Раньше автор приносил рукопись в издательство, затем она попадала в типографию, где ее набирали заново. Кроме лишних затрат труда и времени, при дополнительном наборе в текст вносились дополнительные ошибки. Сегодня во всех издательствах, в редакциях газет и журналов стоят компьютеры. И текст, подготовленный автором на ЭВМ, идет прямо в работу. На компьютере выполняется ***верстка*** — размещение текста на бумаге, после чего текст печатается и передается в типографию для размножения. Таким образом, применение компьютера позволяет отказаться от многократного набора текста, обеспечивая более быструю подготовку изданий с меньшим количеством ошибок.

3.2. Базовые возможности текстового редактора

Сегодня существует огромное количество различных текстовых редакторов. У каждого из них — свои особенности, свои правила работы. Мы опишем некоторые общие принципы, характерные для всех или хотя бы для многих редакторов, а на практических занятиях вы познакомитесь с тем, как работает конкретный редактор.

Ввод текста. Это основная операция, позволяющая создавать новые тексты. Текст просто печатается с использованием клавиатуры компьютера.

Вставка и удаление. При вводе текста неизбежны ошибки. Можно пропустить букву или даже слово либо, наоборот, напечатать лишнее. В любом текстовом редакторе есть возможность удалить из текста символ или группу символов и вставить новые символы в любое место текста.

Удаление символов производится одинаково практически во всех редакторах, а вставка делается по-разному. Существует два основных режима — ***вставка*** и ***замена***. Они отличаются тем, что происходит, если набрать новый символ в середине уже существующего текста.

В режиме замены в этом случае символ, на котором стоит курсор, стирается, а на его месте появляется набранный символ. Это очень удобно, когда надо исправить неверно набранную букву или заменить одно слово другим. Чтобы вставить новый символ (или группу символов) в режиме замены, надо сначала раздвинуть текст (обычно это действие выполняется специальной командой), а затем вписать нужные символы на освободившееся место.

В режиме вставки новый символ раздвигает ранее введенные. Это позволяет вставлять символы без предварительной подготовки места для вставки. А вот выполнять замену в режиме вставки не очень удобно: надо сначала удалить ненужные символы, а затем вставить новые.

Некоторые редакторы позволяют человеку самому выбирать режим вставки или замены и менять его во время работы. В других режим жестко установлен, и изменить его нельзя.

Копирование и перемещение. При работе с текстами часто возникает необходимость повторить какой-то фрагмент (напри-

мер, длинное название) несколько раз или перенести его в другое место. В большинстве редакторов есть возможность отметить (выделить, запомнить) фрагмент текста, а затем повторить (скопировать, вспомнить) его в другом месте. Если при этом выделенный фрагмент остается также и на исходном месте, такое действие называют копированием, а когда он удаляется с исходного места, речь идет о перемещении.

Поиск и замена. При работе с большим текстом не всегда легко быстро найти в нем нужное место. А если с текстом работает не его автор, проблема становится особенно трудной. Обычно в текстовом редакторе есть функция поиска: человек вводит слово (несколько слов, в общем случае — произвольный набор символов), и редактор находит его в тексте (если, конечно, оно там есть).

Еще одна возможность, тесно связанная с поиском, — контекстная замена. Редактор заменяет все вхождения в текст одного набора символов на другой. Например, можно везде вместо слова «ЭВМ» поставить «компьютер». Многие редакторы разрешают делать сплошную или выборочную замену. При сплошной замене заменяются все найденные сочетания. При выборочной, найдя очередное сочетание, редактор показывает соответствующий фрагмент текста и спрашивает, надо ли делать замену в этом месте.

Форматирование. Вы, наверное, обращали внимание, что в книгах (например, в нашем учебнике) правый край текста обычно бывает ровным. При письме от руки или на машинке достичь такого эффекта не удается, а многие редакторы позволяют получить ровный правый край. Эта операция называется форматированием. При форматировании редактор с помощью различных средств (например, вставляя в текст дополнительные пробелы) обеспечивает, чтобы все строчки были одинаковой длины.

В реальных редакторах можно выбрать различные режимы форматирования: например, отказаться от ровного правого края или, наоборот, прижать текст вправо, получив рваный левый край; указать, нужна или нет красная строка, и т. д.

Некоторые редакторы при форматировании сами переносят слова. Поскольку в каждом языке свои правила переноса, в современных редакторах можно указать, на каком языке пишется текст, и тогда, если правила переноса для этого языка известны, редактор сумеет правильно переносить слова.

3.3. Проверка орфографии

Многие современные редакторы имеют возможность проверки орфографии (по-английски *spellchecking*). Редактор просматривает текст, указывает на слова, которые он считает подозрительными, и предлагает свои варианты их написания. Человек либо вносит исправления (если это действительно ошибка), либо оставляет все как есть.

Проверка орфографии может быть устроена разными способами. Простейший из них — сравнение со словарем. В этом случае компьютер хранит словарь языка и ищет в нем каждое слово текста. Если слова в словаре нет, оно считается подозрительным и его предлагается исправить.

Обратите внимание, что при таком способе мы вынуждены включать в словарь все формы каждого слова: все падежи существительных и прилагательных, все формы глаголов и т. д. Это особенно неудобно при расширении словаря. Если, например, в словаре нет слова «алгоритм» (в последующих главах нашего учебника оно будет встречаться очень часто!) и мы добавили его туда, редактор все равно будет останавливаться на словах «алгоритмы», «алгоритму» и т. д., каждую из этих форм придется заносить в словарь как отдельное слово.

В результате объем словаря резко увеличивается, а проверка все равно остается не очень эффективной, так как многие ошибки не будут найдены. Если употребить неправильное в конкретном случае, но имеющееся в словаре слово, это останется незамеченным. Например, если ошибочно написать слово «длина» с двумя «н» — «длинна» (ошибка, которую часто делают школьники!), то получится правильное краткое прилагательное и ошибка не будет найдена.

Значительно более сложный, но и более эффективный способ проверки заключается в грамматическом анализе текста. Здесь нельзя обойтись простым сравнением слов, поэтому к редактору подключаются специальные программы, отдельно для каждого языка.

Обычно такие системы не только находят слова в словаре (совсем без словаря обойтись нельзя), но и узнают части речи, по типичным окончаниям определяют формы слов, анализируют грамматическую структуру предложений. Все это — довольно сложные задачи, над решением которых активно работают ученые-лингвисты. Подобные системы могут находить непра-

вильно построенные фразы, пропущенные слова, ошибки в расстановке знаков препинания.

Понятно, что никакая, даже самая лучшая, программа проверки никогда не сумеет найти *все* ошибки. Ведь для этого необходимо понимать смысл слов и знать, что именно хотел сказать автор.

Например, сочетания «веселая компания» и «веселая кампания» (а в последнее время еще и «веселая КВАмпания») вполне возможны в русском тексте, и если человек ошибочно написал не ту букву, то компьютер вряд ли сумеет это заметить.

Возможны и более тонкие случаи. Например, одна очень хорошая программа проверки текстов на русском языке, находящая более 90% всех типичных ошибок, при анализе предложения, в котором встретилось словосочетание «гибкому стану», сообщила, что к прилагательному «гибкому» нет существительного, а перед глаголом «стану» пропущена запятая.

Так что никакие средства автоматической проверки не могут заменить человеку грамотности и хорошего знания родного языка.

3.4. Простой формат сохранения текста

Подготовленный с помощью редактора текст необходимо сохранить, т. е. записать в виде файла на диск компьютера. Как мы уже знаем, для хранения в памяти (оперативной или внешней) текст, как и любая информация, должен быть закодирован двоичным кодом.

Один из способов двоичного кодирования текста рассмотрен в § 1. Каждый символ кодируется одним байтом, что позволяет иметь 256 различных символов. Этого достаточно для всех заглавных и строчных букв русского и латинского алфавитов, цифр, основных знаков препинания и специальных символов. Таким образом, текст хранится как простая последовательность кодов символов.

Реальные тексты записываются не на телеграфной ленте, а на листе бумаги, поэтому необходимо еще учесть информацию о разбиении на строки. Обычно в конце каждой строки записывается специальный служебный символ, означающий переход на новую строку.

Такой способ представления текста называется ***простым форматом*** (по-английски *plain text*).

3.5. Специальный формат

При оформлении текста часто используют различные выразительные средства: крупный и мелкий шрифты, курсив, различные способы оформления абзацев.

Во многих редакторах можно применять эти средства. Текст при этом получается более красивым, его легче читать.

Некоторые редакторы позволяют очень подробно указать, как именно должен выглядеть текст, отпечатанный на бумаге. Если текст на экране имеет такой же вид, как на бумаге, говорят, что редактор работает по принципу WYSIWYG (аббревиатура английской фразы — ***what you see is what you get*** — что вы видите, то вы и имеете).

Простой формат не позволяет включить в текст дополнительную информацию, поэтому редакторы, использующие эти возможности, сохраняют текст посредством различных специальных форматов. В этих форматах текст содержит не только печатные символы, но и указания о том, каким шрифтом их печатать, как форматировать абзацы, и т. д.

Возможности дополнительного оформления у разных редакторов отличаются, поэтому и правила записи текста в специальном формате у каждого редактора свои. В результате платой за удобство становится несовместимость: текст, подготовленный одним редактором, часто нельзя нормально прочитать в другом, так как эти редакторы используют разные форматы.

3.6. Описательный формат

Достоинство простого формата — его независимость от особенностей конкретного редактора, достоинство специального формата — более развитые возможности оформления текста.

Совместить и то и другое можно в описательном формате. При пользовании описательным форматом в редакторе создается простой текст, в который кроме собственно текста включаются также специальные команды, описывающие, как этот текст должен быть оформлен. Затем текст читается специальной программой, которая выводит его на экран или на бумагу в соответствии с этими указаниями.

Описательный формат позволяет готовить тексты с очень высоким качеством оформления в самых простых редакторах и даже на слабых компьютерах. Полученные тексты занимают

сравнительно немного места, ими легко обмениваться. Человек, знакомый с соответствующим форматом, может читать такой текст непосредственно, без помощи специальной программы чтения и преобразования. Но надо понимать, что идеальных решений не существует. Платой за удобства описательного формата является отказ от принципа WYSIWYG.

Описательных форматов существует довольно много. На сегодняшний день три таких формата получили очень широкое распространение и стали признанными мировыми стандартами.

Первый из них — это **формат RTF** (аббревиатура английских слов ***Rich Text Format*** — обогащенный текстовый формат). RTF — очень громоздкий формат, служебная информация в нем занимает объем, в несколько раз больший, чем собственно текст, поэтому читать и писать текст в RTF очень неудобно. Зато многие редакторы, использующие специальные форматы, могут автоматически преобразовать текст в RTF и обратно. Таким образом, этот формат применяется для преобразования текста из одного специального формата в другой.

Пример текста в формате RTF вы можете увидеть в таблице 6.

ТАБЛИЦА 6. **Пример подготовки текста в формате RTF**

Описание текста в формате RTF

```
1   {\rtf1
2   {\fonttbl
3   {\f0 Courier New;}
4   {\f1 Times New Roman Cyr;}
5   }
6   {\colortbl\red0\green0\bleu0;}
7   \plain\f0\fs20 RTF text \par
8   \plain\f0\fs30 \b RTF text bold \par
9   \plain\f0\fs40 \i RTF text italic \par
10  \plain\f0\fs60\RTF \f1\'f2\'e5\'ea\'f1\'f2\par
11  \plain\f1 \fs40
12  \'e0\'e1\'e2\'e3\'e4\'e5\'e6\'e7\'e8\'e9\'ea\'eb\'ec\'ed\'ee\'ef
13  \'f0\'f1\'f2\'f3\'f4\'f5\'f6\'f7\'f8\'f9\'fa\'fb\'fc\'fd\'fe\'ff\par
14  }
```


Изображение текста на экране

RTF text
RTF text bold
RTF text italic
RTF текст
абвгдежзийклмноп рстуфхчшщъыьэюя

На *экране* мы видим пять строк текста, написанных «по-английски» и «по-русски» разными стилями и размерами. Задание этих пяти строк в описательном формате занимает 14 строк, записанных только буквами латинского алфавита, цифрами, несколькими знаками препинания и спецзнаками: точка с запятой, фигурные скобки и т. д. Давайте «расшифруем» эту запись.

В строке 1 сообщается, что начинается запись текста в формате RTF. В последней строке 14 говорится, что текст закончился (эта строка содержит закрывающую скобку, парную к открывающей скобке строки 1). Строки 2—5 — это так называемая таблица шрифтов (англ. — fonts table). Эта таблица говорит, что текст будет написан двумя шрифтами. Краткие обозначения этих шрифтов f0, f1. Строка 6 — таблица цветов. В ней указано, что красный (red), зеленый (green) и синий (blue) цвета имеют яркость 0. Это означает, что все цвета выключены, т. е. текст написан черным цветом.

Далее в строках 7—13 идет смесь из собственно текста (*что* должно быть изображено на экране) и команд (*как* текст надо изобразить). Вот примеры команд:

\f0 — переключиться на шрифт f0;
\f1 — переключиться на шрифт f1;
\fs20 — переключиться на размер шрифта в 20 условных единиц;
\b — переключиться на **полужирное** начертание;
\i — переключиться на *курсивное* начертание;
\plain — переключиться на обычное начертание (отменить \b и \i);
\par — перейти на новую строку;
\'e0 — вывести символ текущего алфавита с 16-ричным кодом e0.

Расшифруем строку 7:

\plain\f0\fs20 RTF text\par

Эта строка говорит, что нужно изобразить на экране текст "RTF text" шрифтом f0 размера 20 без полужирного начертания и курсива, а следующий текст следует писать с новой строки. Строка 8 сообщает, что нужно изобразить на экране текст "RTF text bold" шрифтом f0 размера 30, используя полужирное начертание. Чуть сложнее устроена строка 10. Она говорит, что сначала нужно изобразить слово "RTF" шрифтом f0 размера 60, а потом в той же строке написать несколько символов шрифта f1. Шрифт f1, т. е. Times New Roman Cyr, включает в себя кириллицу, и в строке указаны коды пяти русских букв:

'f2 — 16-ричный код буквы т;
'e5 — 16-ричный код буквы е;
'ea — 16-ричный код буквы к;
'f1 — 16-ричный код буквы с;
'f2 — 16-ричный код буквы т.

Строки 11—13 содержат коды 32 букв русского алфавита, разделенных в середине несколькими пробелами.

Второй общепринятый **формат** называется **TeX**. Этот стандарт предназначен в первую очередь для математиков, так как он позволяет очень удобно записывать сложные математические формулы. Ниже представлена формула нахождения корней квадратного уравнения (в курсе математики вы познакомитесь с ней в 8 классе) в обычной математической записи:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a};$$

и в формате TeX:

$$ x_{1, 2} = \frac{-b\pm\sqrt{b^2 - 4ac}}{2a} $$

Еще один описательный **формат — HTML** (аббревиатура английских слов ***Hyper Text Markup Language*** — язык разметки гипертекстов). В этом формате записываются тексты, размещаемые во всемирной компьютерной сети Интернет. С ростом Интернет он становится все более популярным, так что мы рассмотрим его подробнее.

3.7. Основные элементы HTML

HTML — описательный формат, поэтому в текст включается информация об оформлении. Эта информация записывается в виде специальных меток, которые называются ***тегами***. Чтобы отличить их от текста, теги заключают в угловые скобки (знаки «меньше» и «больше»).

Ниже приводится пример простейшего текста в HTML и вид этого текста после обработки (рис. 6).

```
<html1>
<head> <title>Авиабилеты</title> </head>
<body

<p> <font size= "5" face="Arial">
Минимальные цены авиакомпании <b>Аэрофлот</b> : <br> <font> </p>

    <p align="center">
    Нью-Йорк - 665 у.е.<br>
    Вашингтон - 654 у.е.<br>
    Сан-Франциско - 844 у.е.<br> </p>

<p> Если у Вас перелеты внутри страны, наши операторы подберут Вам
<u><i>оптимальный </i></u> вариант. </p>

<p> <a href="avia_b.htm">ЗАБРОНИРОВАТЬ АВИАБИЛЕТ </a> </p>
</body>
```

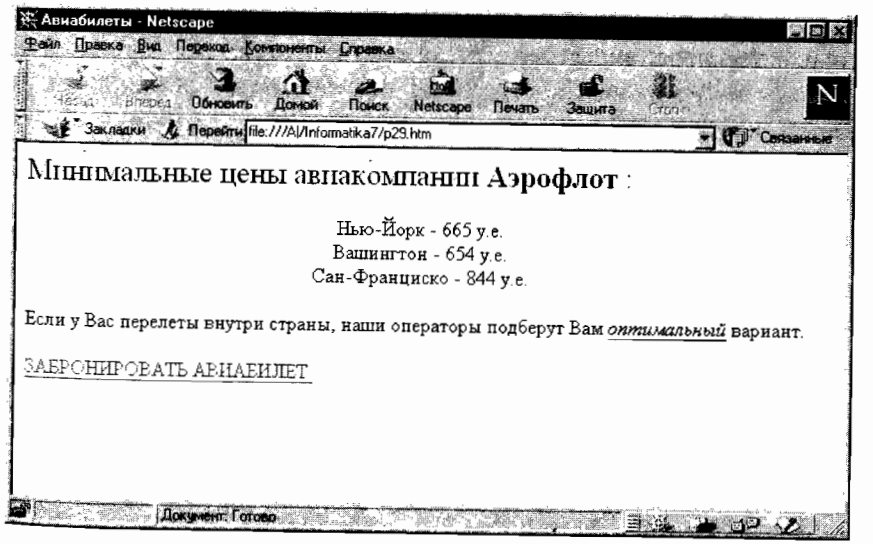


Рис. 6

Теги записываются латинскими буквами, причем строчные и заглавные буквы не различаются. Например, <HTML> и <html> — это один и тот же тег.

Теги содержат указания о том, как должен выглядеть текст. Многие теги встречаются парами: открывающий и закрывающий. Они обозначаются одним и тем же словом, но перед закрывающим добавляется знак «/» — косая черта. Открывающий тег означает включение какого-то режима, закрывающий — его выключение.

Текст в формате HTML принято называть ***страницей***. Обычно страница имеет такую структуру:

```
<HTML>
   <HEAD>
         специальная информация
   </HEAD>
   <BODY>
         собственно текст
   </BODY>
</HTML>
```

Весь текст расположен между тегами <HTML> и </HTML>.

Текст делится на две части. Между тегами <HEAD> и </HEAD> располагается специальная служебная информация. Чаще всего там указываются специальные параметры, необходимые для передачи страницы по сети Интернет. Один из таких параметров — название страницы, оно заключается между тегами <TITLE> и </TITLE>.

Вторая часть помещается между тегами <BODY> и </BODY>. Здесь находится собственно текст страницы. В этот текст часто входят различные теги, описывающие его оформление.

Некоторые теги содержат ***параметры***, т. е. дополнительную информацию, уточняющую действие тега. Например, тег <FONT>, определяющий шрифт текста, включает в себя дополнительный параметр SIZE, указывающий размер символов.

Размер символов в HTML задается числом от 1 до 7. 1 — очень мелкий шрифт, 7 — очень крупный. Если размер не задан, то обычно считается, что он равен 4 — среднему из возможных значений. С помощью тега <FONT> этот размер можно изменить.

Тег <FONT> позволяет задавать абсолютный и относительный размеры. Абсолютный размер непосредственно указывает

величину шрифта. Например, <FONT SIZE = 2> означает, что дальнейший текст будет выводиться мелким шрифтом (размер 2). Относительный размер задается со знаком («плюс» или «минус») и показывает величину, которую надо прибавить к текущему размеру шрифта. Например, если весь текст имеет размер 4, после тега <FONT SIZE = +2> будет включен размер 6.

При использовании различных комбинаций тегов необходимо соблюдать правильную ***вложенность*** открывающих и закрывающих тегов. Тег, который был открыт *позже*, должен быть закрыт *раньше*. Вот пример правильной последовательности: <B><I> жирный курсив </I></B>. А вот последовательность <B><I> жирный курсив </B></I> нарушает правила вложенности, результат ее обработки может оказаться ошибочным.

Основные теги HTML, необходимые для создания простых текстов, приведены в таблице 7. Использованный на рисунке 6 тег <a...>, позволяющий организовать гиперссылки на другие тексты в Интернете, описан в разделе 37.4.

ТАБЛИЦА 7. **Некоторые теги HTML**

Открывающий тег	Закрывающий тег	Комментарий
<HTML>	</HTML>	Текст в формате HTML
<HEAD>	</HEAD>	Служебная часть текста
<TITLE>	</TITLE>	Название страницы (помещается в служебной части; не входит в текст; видно при просмотре страницы в сети Интернет)
<BODY>	</BODY>	Основная часть текста
<H1>	</H1>	Основной заголовок
<H2>	</H2>	Подзаголовок
<P>	</P>	Абзац текста
<B>	</B>	Жирный шрифт
<I>	</I>	Курсив
<U>	</U>	Подчеркивание
<FONT SIZE =n>	</FONT>	Изменение размера шрифта

ЗАДАЧИ И УПРАЖНЕНИЯ

1. При автоматической проверке орфографии очень важно правильно определить часть речи, к которой относится каждое слово. Объясните проблемы, которые возникают при анализе следующих слов: *пила, устав, мой, три, простой, слив, пасть*.
2. Сформулируйте общее свойство слов, перечисленных в упражнении 1. Придумайте свои слова с этим свойством.
3. По заданному изображению страницы (рис. 7) восстановите исходный текст в HTML.

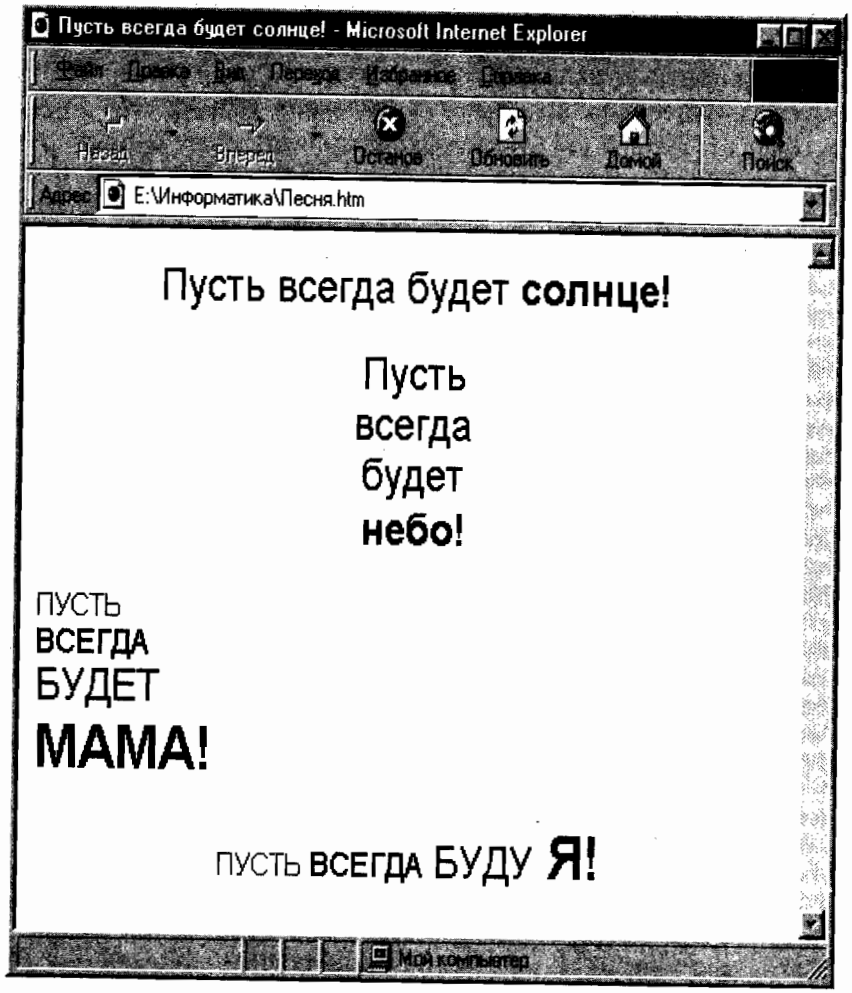


Рис. 7

4 Информационные системы и базы данных

4.1. Хранение и поиск информации

Любая деятельность человека всегда связана с информацией. В самом деле, ведь прежде чем делать что-либо, мы должны *знать*, что мы собираемся делать, а знание — это уже информация.

В предыдущем параграфе мы обсуждали проблемы, связанные с ***подготовкой*** информации, представленной в виде текстов. Теперь мы обсудим вопросы ее хранения и поиска.

Чтобы информацией можно было воспользоваться, она должна быть сохранена. Человек постоянно применяет многочисленные системы хранения информации — от персональной записной книжки до огромных библиотек.

Но сохранить информацию недостаточно. Она должна быть сохранена так, чтобы при необходимости ее можно было быстро найти, иначе в подобном хранении нет никакого смысла. Именно для упрощения поиска люди придумали алфавитные записные книжки, а библиотеки создают подробные каталоги.

Хранение и поиск информации необходимы везде. В любой организации, на любом предприятии обязательно ведется работа с документами. Со временем количество этих документов растет, и, чтобы не запутаться в них, применяют специальные системы делопроизводства, позволяющие упорядочить информацию.

Очень важную роль играет поиск информации в научной деятельности. При недостатке информации ученые дублируют работу друг друга, повторяют уже выполненные кем-то исследования.

Во всех перечисленных и во многих других случаях необходимо так организовать хранение информации, чтобы ее можно было легко и быстро найти.

До появления компьютеров для этой цели использовались различные каталоги и картотеки. Развивалась даже специальная наука, которая изучала различные средства упорядочения и поиска информации. Название этой науки может показаться вам неожиданным: ***информатика***.

С появлением и развитием универсальных информационных машин — компьютеров — слово ***информатика*** несколько изменило свой смысл. А хранение и поиск информации все чаще ведется с помощью компьютеров.

4.2. Пример информационной задачи

Автоинспекция ведет регистрацию автомобилей. В специальной картотеке хранится информация о каждом автомобиле: номер машины, марка, цвет, данные о владельце. Все данные в картотеке упорядочены по номерам машин, поэтому сведения о любой машине легко найти, если известен ее номер.

К сожалению, номер известен не всегда. Бывают, например, случаи, когда водитель уезжает с места происшествия, а свидетели в лучшем случае называют только марку и цвет, но не помнят номера. Чтобы найти, например, все белые «Москвичи», приходится перебирать всю картотеку — огромная работа, которой еще не так давно действительно приходилось заниматься.

Конечно, можно завести несколько разных картотек: в одной упорядочить карточки по номерам, в другой — по цвету, в третьей — еще по какому-то признаку. Но такое дублирование значительно увеличивает объем и усложняет работу по созданию и пополнению картотеки, поскольку каждую карточку надо делать в нескольких экземплярах. Кроме того, в реальных информационных системах хранится множество различных данных, и не всегда можно заранее предсказать, по какому признаку придется вести поиск.

Решение подобных информационных задач, обеспечивающее поиск любой имеющейся информации за приемлемое время, стало возможно только с появлением компьютеров.

4.3. Табличная база данных

Для хранения и поиска информации применяют специальные компьютерные системы, которые называются ***базы данных*** (сокращенно — БД). Мы будем изучать работу с БД на примере самых простых баз данных — ***табличных***.

Чтобы получить наглядное представление о табличной БД, занесем в таблицу информацию о нескольких автомобилях, зарегистрированных в автоинспекции, о которой говорилось выше.

ТАБЛИЦА 8. **Информационная система**

Номер	Марка	Цвет	Выпуск	Регистрация	Владелец
А123АА77	«Жигули»	Синий	1990	1994	Иванов
В003ВВ77	«Лорен—Дитрих»	Зеленый	1925	1999	Козлевич
Н380ХУ77	«Форд»	Бежевый	1992	1998	Кушниренко
С487ОР77	«Волга»	Черный	1998	1998	Мюллер
У910АЕ99	«Мерседес»	Красный	1999	2000	Козлов

Таблица включает в себя строки, каждая из которых описывает данные об одном автомобиле. В базах данных строки принято называть ***записями***. Таким образом, база данных состоит из записей.

Каждая графа (столбец) таблицы содержит описание какой-то характеристики автомобиля. В базах данных эти графы называют ***полями***. Таким образом, каждая запись состоит из полей.

В таблице 8 каждая запись состоит из 6 полей. (В реальной базе данных автоинспекции есть и другие поля, например **мощность**, которые мы для упрощения не рассматриваем.)

4.4. Действия с базами данных

Основные действия при работе с базами данных — это проектирование, создание, поддержка, поиск и обработка.

Прежде чем работать с базой данных, ее надо ***создать***, т. е. ввести в компьютер форму таблицы и заполнить ее информацией.

Но чтобы создать БД, ее сначала надо ***спроектировать***, т. е. определить, какие именно поля будут в таблице с данными, в каком порядке они будут располагаться. Проектирование — очень важный этап, так как поменять список полей в уже заполненной таблице в реальных базах данных иногда бывает сложно.

При работе с базой данных информация может изменяться. Например, в нашей учебной автомобильной базе данных появляются новые машины, какие-то записи необходимо исключить (машина снята с учета), а в каких-то уточнить информацию (машина перешла к другому владельцу или просто перекрашена). Эти операции называют ***поддержкой*** базы данных. Поддержка включает в себя ***добавление*** новых записей, ***удаление*** записей и ***исправление*** информации в существующих записях.

Наконец, самые важные операции — это ***поиск*** информации в базе данных и ее ***обработка***. Поиск позволяет среди множества записей (а в реальных БД записей может быть очень много) выделить нужные, обработка — получить новую информацию на основе данных, которые хранятся в БД.

Создание и поддержка базы данных — операции чисто технические, чтобы выполнять их, достаточно разобраться в соответствующих компьютерных программах.

Для информатики особый интерес представляют проектирование, поиск и обработка информации.

4.5. Проектирование базы данных

При проектировании базы данных сначала следует определить, какой ***объект*** будет описываться каждой записью. Например, объект в базе данных из таблицы 8 — автомобиль.

Затем надо решить, какая информация об объекте будет храниться в БД, и представить эту информацию в виде списка полей. Каждое поле необходимо *описать*, т. е. указать его ***имя*** и ***тип***.

Имя поля — это его название, заголовок графы в таблице. Это имя потом будет использоваться во всех остальных операциях, поэтому надо позаботиться, чтобы оно было понятным, но при этом не слишком длинным. Имя может состоять из букв и цифр, но обязательно должно начинаться с буквы.

Тип поля указывает, какие *значения* может принимать информация в этом поле и какие действия можно выполнять с ним при поиске и обработке.

Мы будем использовать три типа полей: ***целое, вещественное*** и ***литерное***.

Целое поле служит для хранения целого числа. Вещественное — для любого (не обязательно целого) числа. Необходимость различать эти типы связана с тем, что в компьютере обработка целых и вещественных чисел происходит по-разному.

В литерном поле может содержаться любой набор символов, это текстовое (не числовое) поле.

Мы будем описывать спроектированную базу данных специальной формы. Рассмотрим ее на примере уже знакомой нам базы данных автомобилей. Ее можно описать так:

```
запись автомобиль
    лит   номер        | регистрационный номер
    лит   марка
    лит   цвет
    цел   выпуск       | год выпуска автомобиля
    цел   регистрация  | год регистрации
    лит   владелец     | фамилия владельца
конец записи
```

Слова **запись** и **конец записи** отмечают начало и конец описания записи. Слова **лит** и **цел** обозначают поля литерного и целого типов. Для поля вещественного типа (в нашем примере их нет) используется слово **вещ**. Все эти слова называются ***служебными***. Это означает, что их смысл строго определен, и ни в каком другом смысле употреблять эти слова при работе с базами данных нельзя.

Слово *автомобиль* в этом примере — имя всей записи в целом, слова *номер, марка, цвет, выпуск, регистрация, владелец* — имена отдельных полей.

После вертикальной черты записываются ***комментарии***. Это дополнительные пояснения, которые не являются частью описания, но помогают человеку (не компьютеру!) лучше понять смысл написанного.

4.6. Поиск в базе данных

Задание на поиск информации в базе данных называется ***запросом***. Запросы записываются по специальным правилам, которые называются ***языком запросов***. Языки запросов в разных системах могут отличаться, но в их основе обычно лежат одни и те же общие принципы. Мы будем пользоваться простым учебным языком запросов. Не описывая строгих правил этого языка, просто разберем несколько примеров.

1. Сравнение литерных значений. Чтобы найти все автомобили синего цвета, достаточно ввести в компьютер такой запрос:

```
цвет="синий"
```

Запрос для поиска машины, принадлежащей Кушниренко, будет выглядеть так:

```
владелец="Кушниренко"
```

В случае полей литерного типа значение записывается в кавычках.

2. Сравнение числовых значений. Найдем все автомобили, выпущенные в 1998 г.

Запрос будет иметь вид:

```
выпуск=1998
```

Числа можно сравнивать не только на равенство. При поиске автомобилей, выпущенных ранее 1985 г., удобно дать такой запрос:

```
выпуск<1985
```

Можно задавать и двойные неравенства. Найдем автомобили, выпущенные с 1989 по 1994 г. включительно:

```
1989<=выпуск<=1994
```

3. Сравнение полей между собой. В запросе допускается сравнивать между собой значения различных полей. Для поиска автомобилей, зарегистрированных в год выпуска, дадим, например, такой запрос:

```
выпуск=регистрация
```

Иногда стоит попробовать найти машины, зарегистрированные раньше, чем они были сделаны:

```
регистрация<выпуск
```

Понятно, что на самом деле такого быть не может, но если при заполнении БД была допущена ошибка, то подобный запрос поможет ее найти.

4. Использование арифметических операций. В запросы можно включать арифметические операции. Сложение и вычитание обозначаются привычными плюсом и минусом, умножение — звездочкой (*), деление — косой чертой (/).

Вот запрос на поиск автомобилей, от выпуска до регистрации которых прошло больше 3 лет:

```
регистрация—выпуск>3
```


5. Поиск по шаблону. Часто возникают ситуации, когда надо найти литерное поле, значение которого известно не полностью. В этом случае удобно задать поиск с помощью ***шаблонов*** — образцов, которые не указывают точное значение поля, но позволяют отделить допустимые значения.

В шаблонах мы будем использовать два специальных символа: звездочка (*) означает, что на соответствующем месте в значении поля могут находиться любые символы, причем количество их неизвестно (возможно, что их и вообще нет). Вопросительный знак (?) соответствует одному произвольному символу.

Рассмотрим несколько примеров запросов с шаблонами.

Машины, фамилии владельцев которых начинаются с буквы «К»:

```
владелец="К*"
```

Надо найти автомобиль, владелец которого то ли Кошкин, то ли Кашкин, точная фамилия неизвестна. Возможный запрос:

```
владелец="К?шкин"
```

Машины, номера которых состоят ровно из пяти символов:

```
номер="?????"
```

Машины, в номерах которых пять или больше символов:

```
номер="?????*"
```

Машины, номера которых содержат цифру 3:

```
номер="*3*"
```

4.7. Составные запросы

Все запросы, которые мы строили до сих пор, вели поиск по какому-то одному признаку. Для каждой записи в базе данных проводилось единственное сравнение, и по его результатам определялось, надо включать эту запись в результаты запроса или же нет. Такие запросы называются ***простыми***.

Часто возникает необходимость вести поиск сразу по нескольким признакам. Допустим, нам надо найти в списке автомобилей все черные «Волги». Запрос марка="Волга" позволит нам найти все «Волги», но среди них будут не только черные. Аналогично запрос цвет="черный" найдет все черные машины, включая ненужные нам «Жигули» и «мерседесы».

Для решения подобных задач применяются ***составные запросы***. Вот как можно найти все черные «Волги»:

марка="Волга" **и** цвет="черный"

Служебное слово **и**, соединяющее два простых запроса, показывает, что мы ищем те записи, которые удовлетворяют *каждому* из этих запросов.

Рассмотрим еще несколько примеров.

Найти все «Москвичи», выпущенные до 1995 г.:

марка="Москвич" **и** выпуск<1995

Найти автомобили, выпущенные до 1995 г., а зарегистрированные — после него:

выпуск<1995 **и** регистрация>1995

Найти «форды», номера которых начинаются с нуля, а фамилия владельца — с буквы «Б»:

марка="форд" **и** номер="0*" **и** владелец="Б*"

Найти автомобили, в номере которых есть цифры 3 и 7:

номер="*3*" **и** номер="*7*"

Пусть теперь нам надо найти все черные и все синие автомобили. Запрос цвет="черный" **и** цвет="синий" не приведет нас к успеху. Слово **и** показывает, что оба условия должны выполняться *для одной и той же записи*. Поскольку автомобиль не может быть одновременно черным и синим, результатом такого запроса будет пустой список, не содержащий ни одной записи.

Запрос для решения подобной задачи должен выглядеть так:

цвет="черный" **или** цвет="синий"

Служебное слово **или**, соединяющее два запроса, показывает, что подходящими должны считаться все записи, которые удовлетворяют *хотя бы одному* из этих запросов.

Иногда бывает нужно найти все записи, которые *не* удовлетворяют какому-то условию. Предположим, надо найти все автомобили не красного цвета. Это можно сделать с помощью такого простого запроса:

цвет<>"красный"

Но можно поступить по-другому и использовать составной запрос со специальным служебным словом **не**:

не (цвет="красный")

Слово **не** означает, что надо выполнить стоящий после него запрос и выбрать записи, которые запросу *не удовлетворяют*. Иногда с помощью этого слова удается записать запросы более понятно. Например, вот как может выглядеть запрос на поиск всех автомобилей, кроме красных и белых:

не (цвет="красный" **или** цвет="белый")

В сложных случаях приходится строить комбинированные запросы, использующие одновременно операции **и**, **или**, **не**. Для указания порядка действий в таких сложных запросах можно использовать скобки.

Примеры. Найти автомобиль, про который известно, что это «форд» или «джип» черного или синего цвета:

(марка="форд" **или** марка="джип") **и** (цвет="черный" **или** цвет="синий")

Найти «Запорожцы» и «Таврии», зарегистрированные после 1991 г.:

(марка="Запорожец" **или** марка="Таврия") **и** регистрация>1991

4.8. Базы данных и СУБД

Для работы с базами данных на компьютере необходимы специальные программы. Однако создавать отдельную программу для каждой БД было бы очень дорого и неудобно. Ведь у баз данных, даже описывающих совершенно различные, далекие друг от друга предметные области, есть много общего. Например, правила описания структуры базы и язык запросов, которые мы рассмотрели в этом параграфе, никак не связаны с регистрацией автомобилей. Пользуясь этими же правилами, мы можем составить много других баз данных (см. упражнения).

Поэтому сегодня принято разделять собственно базу данных и программу для работы с ней. Программа определяет допустимую структуру информации в БД (напомним, что мы рассматривали только табличную структуру), правила ввода информации, язык запросов. Такая программа называется ***система управления базами данных*** (***СУБД***).

С помощью одной и той же СУБД можно создать множество баз данных, содержащих самую разнообразную информацию.

ЗАДАЧИ И УПРАЖНЕНИЯ

1. Подумайте, какие поля могла бы включать база данных, содержащая сведения:
   а) об учениках вашего класса;
   б) о вашей домашней фонотеке;
   в) о планетах Солнечной системы;
   г) о химических элементах;
   д) о странах мира;
   е) о породах собак;
   ж) о товарах в компьютерном магазине.
2. Составьте формальное описание баз данных из упражнения 1.
3. Запишите 3—4 различных запроса к базе данных, которую вы описали в упражнении 2. Поясните смысл этих запросов.
4. Дан фрагмент описания базы данных:

   запись страна
   ...
   вещ площадь | площадь в тысячах км^2
   вещ население | население в миллионах человек
   ...
   конец записи

   Составьте запросы для нахождения:
   а) стран большой площади (больше 100 тыс. км^2);
   б) стран с большим населением (больше 50 млн чел.);
   в) больших стран (т. е. тех, которые могут считаться большими либо по площади, либо по населению);
   г) очень больших стран (т. е. имеющих и большую площадь, и большое население);
   д) больших, но при этом не очень больших стран;
   е) стран с большой плотностью населения (больше 500 чел./км^2).
5. Дан фрагмент описания базы данных:

   запись элемент
   ...
   лит название | русское название химического элемента
   лит символ | обозначение элемента в формулах
   ...
   конец записи

   Составьте запросы для нахождения элементов:
   а) с названиями из пяти букв;
   б) с названиями на букву «а»;

   в) обозначение которых содержит букву «f» (большую или малую);
   г) в названии которых есть буква «о»;
   д) в названии которых есть буквы «о» и «а»;
   е) в названии которых есть буква «о», но нет «а»;
   ж) в названии которых есть две буквы «о»;
   з) в названии которых ровно две буквы «о».

5 Электронные таблицы

5.1. Компьютер — инструмент вычислений

И русское название «электронная вычислительная машина», и английское слово *computer* (вычислитель) подчеркивают способность компьютера к вычислениям. Действительно, первые компьютеры создавались именно для решения вычислительных задач, и долгое время различные расчеты оставались основным их применением. И сегодня среди многочисленных применений современных компьютеров важное место по-прежнему занимают научные, инженерные, бухгалтерские и другие задачи, в которых требуется быстро и точно считать.

В истории человечества было много попыток создать устройство, способное выполнять вычисления. Абак, счеты, логарифмическая линейка, арифмометр, калькулятор — при всех различиях этих устройств они делают одну и ту же работу: вычисляют результат заданных человеком арифметических действий.

Все названные устройства в состоянии *выполнять* вычисления, но *организация* действий остается за человеком. Пользуясь арифмометром или калькулятором, человек вручную вводит все значения и задает необходимые действия. Если нужно провести одинаковые вычисления с различными исходными данными, все действия приходится повторять заново.

Компьютер дает возможность указать последовательность выполнения действий в автоматическом режиме, без вмешательства человека в процесс вычислений, а также позволяет отделить описание действий от тех данных, с которыми эти действия производятся. Это означает, что можно повторять вычисления с разными исходными данными, не вводя каждый раз заново описание необходимых действий.

Таким образом, появление компьютера привело к качественным изменениям как в решении вычислительных задач, так и в других задачах обработки информации.

5.2. Вычисления и программирование

Как же указать компьютеру, какие действия надо выполнить? Один из способов сделать это — составить программу. Работая по программе, компьютер запрашивает у человека исходные данные (или получает их каким-то другим способом, например, вводит из файла или снимает показания с датчиков), выполняет заданные в программе вычисления и сообщает результат. Можно повторить вычисления, заменив исходные данные, и при этом заново писать программу не нужно.

Но когда изменяются правила вычислений, приходится менять программу, а для этого требуется специальная квалификация, нужен программист. Кроме того, программирование обычно требует времени, исправление и проверка программ — не мгновенная операция.

Получается, что организация вычислений с помощью программирования дает существенный выигрыш по сравнению с калькулятором лишь в том случае, если расчетные формулы жестко заданы и практически не подвержены изменениям. Если же формулы необходимо менять, калькулятор оказывается более гибким средством.

5.3. Электронные таблицы

Ситуация коренным образом изменилась после изобретения электронных таблиц — особого вида программ, которые позволяют человеку, не прибегая к программированию, одинаково легко задавать и изменять как числовые данные, так и формулы для вычислений.

Работая с электронными таблицами, человек видит на экране компьютера таблицу, состоящую из строк и столбцов. В клетках этой таблицы записываются числа. Подобная форма часто встречается в традиционной некомпьютерной обработке данных: в табличном виде обычно представляются бухгалтерские данные, результаты научных экспериментов, статистические отчеты.

Вся информация в электронных таблицах делится на ***исходную*** и ***вычисляемую***. Исходная информация заносится в таблицы человеком, а вычисляемая рассчитывается автоматически. Главный принцип работы электронных таблиц — это возможность ввода и редактирования ***формул***, которые задают правила определения вычисляемой информации на основе исходной.

Для работы с электронными таблицами есть много различных программ. Разберем пример использования в бухгалтерии учреждения одной такой программы — редактора электронных таблиц «Микрокальк».

Пользуясь программой, бухгалтер заносит в ведомость на начисление зарплаты по отделу 6 *исходные* данные: графы "Начислено" и "Кол-во минимальных".

Ведомость на выдачу зарплаты за март месяц 2000 года по отделу № 6 ООО "Иванов и компания"

ФИО	Начислено	Удержано в ПФ	Кол-во минимумов	Облагаемый доход	Подоходный налог	На руки
Иванов	3300		2			
Иванова	2000		3			
Петров	2500		2			
Петрова	600		2			
Сидоров	1200		2			
ИТОГО			=	=		

ММРОТ = 83.49 ! минимальный месячный размер оплаты труда
Удержано в ПФ = Начислено * 1%
Облагаемый доход = Начислено – Удержано в ПФ – Кол-во минимумов * ММРОТ
Подоходный налог = Облагаемый доход * 12%
На руки = Начислено – Удержано в ПФ – Подоходный налог

Ничего больше делать бухгалтеру не придется: по нажатии на специальную клавишу "Микрокальк" проведет все предусмот-

ренные таблицей вычисления и поместит вычисленные результаты в нужные клетки таблицы.

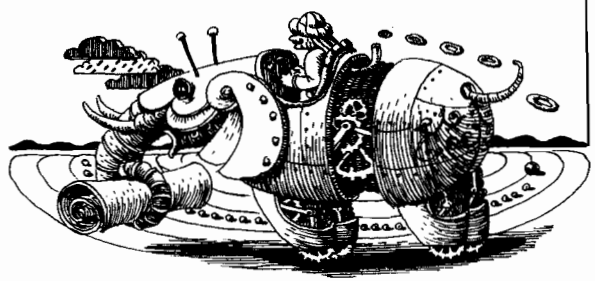
Ведомость на выдачу зарплаты за март месяц 2000 года по отделу № 6 ООО "Иванов и компания"						
ФИО	Начислено	Удержано в ПФ	Кол-во минимумов	Облагаемый доход	Подоходный налог	На руки
Иванов	3300	33.00	2	3 100.02	372.00	2 895.00
Иванова	2000	20.00	3	1 813.02	217.56	1 762.44
Петров	2500	25.00	2	2 308.02	276.96	2 198.04
Петрова	600	6.00	2	427.02	51.24	542.76
Сидоров	1200	12.00	2	1 021.02	122.52	1 065.48
ИТОГО	9600	96.00	=	=	1 040.28	8 463.72

ММРОТ = 83.49 ! минимальный месячный размер оплаты труда
Удержано в ПФ = Начислено * 1%
Облагаемый доход = Начислено – Удержано в ПФ – Кол-во минимумов * ММРОТ
Подоходный налог = Облагаемый доход * 12%
На руки = Начислено – Удержано в ПФ – Подоходный налог

Ведомость готова. Ее можно распечатать без формул в нижней части таблицы (для передачи кассиру) либо целиком (для сдачи в архив). Формулы в нижней части таблицы составлены с учетом действующего законодательства. Например, в марте 2000 г. минимальный месячный размер оплаты труда составлял 83 р. 49 к., в пенсионный фонд удерживался 1%, а подоходный налог для доходов менее 50 тыс. р. в год составлял 12%.

ГЛАВА 2

Исполнители и алгоритмы



6 Исполнители

6.1. Исполнители вокруг нас

Современного человека окружает множество разнообразных технических устройств. Телевизор, фотоаппарат, стиральная машина, магнитофон, автомобиль... Этот список можно продолжать практически бесконечно. У каждого устройства — свое предназначение, свои особенности, свои правила пользования. Но всеми этими устройствами управляет человек, и управляет по общей схеме. С помощью пульта управления он задает необходимые действия, и устройство выполняет их.

Пульт управления может выглядеть по-разному, но обычно он состоит из различных кнопок, рычагов, регуляторов. Различны и действия, выполняемые устройствами. Нас же будут интересовать не различия, а одно важное общее свойство: у каждого устройства есть строго заданный ограниченный набор возможных действий и каждому действию соответствует какая-то операция с пультом управления.

Устройство, способное выполнять определенный набор команд, мы будем называть ***исполнителем***. Команды, которые

может выполнить конкретный исполнитель, образуют ***систему команд исполнителя***.

Пример. Исполнитель *Магнитофон*.

Система команд:
- начать перемотку вперед
- начать перемотку назад
- начать воспроизведение
- начать запись
- стоп
- установить громкость
- установить уровень записи

Примечание. Составить *полную* систему команд *реальных* технических устройств довольно сложно, поэтому здесь и в ряде последующих примеров приводится только часть возможных команд.

Для работы с исполнителем не обязательно понимать, как выполняется та или иная команда, достаточно знать, *что* сделает исполнитель. Например, чтобы пользоваться магнитофоном, не нужно разбираться в устройстве лентопротяжного механизма, достаточно знать, что при нажатии соответствующей кнопки лента будет перематываться.

6.2. Состояния исполнителя

Состояние каждого исполнителя во время работы можно описать какими-то характеристиками. Например, состояние исполнителя *Телевизор* описывают выбранные канал, громкость, яркость, контрастность и т. д.

Состояние исполнителя иногда меняется при выполнении команд (например, при настройке телевизора) или с течением времени (время, оставшееся до звонка таймера).

Полный набор характеристик, описывающих состояние исполнителя, называется ***средой*** этого исполнителя.

6.3. Команды-приказы и команды-вопросы

При работе с исполнителями необходимо управлять их состоянием, добиваясь выполнения нужных действий. Для этого существуют ***команды-приказы***, к которым относится большинство команд исполнителей. Всевозможные команды управления, переключения, настройки — все это команды-приказы.

Иногда возникает необходимость узнать какие-то характеристики, не изменяя их. Для этого существуют ***команды-вопросы***. При выполнении команды-вопроса состояние исполнителя остается прежним, он только сообщает информацию о нем.

Информация о состоянии реальных устройств часто доступна непосредственно, без выполнения специальных команд. Например, чтобы узнать скорость автомобиля, достаточно просто посмотреть на спидометр.

Иногда команду-вопрос необходимо явно передавать исполнителю. Скажем, современные модели телевизоров показывают текущий номер канала при нажатии специальной клавиши на пульте управления.

6.4. Непосредственное и программное управление

В быту управление устройствами-исполнителями происходит чаще всего по такой схеме: человек дает исполнителю команду, исполнитель выполняет ее, человек смотрит на результат и дает следующую команду и т. д. При этом человек узнает результат выполнения каждой команды (возможно, с помощью команд-вопросов), и выбор следующей команды может зависеть от текущего состояния исполнителя.Такая система называется ***непосредственным управлением*** (рис. 8).

Рис. 8

Однако часто возникают ситуации, когда непосредственное управление неудобно или даже неприменимо.

Пример 1. *Стиральная машина.* Процесс стирки обычно включает в себя стандартную последовательность действий: замачивание, отстирывание, полоскание, сушка. В современных стиральных машинах с программным управлением все эти действия автоматически выполняются в необходимой последовательности без участия и контроля человека. Как правило, в такие машины заложено несколько вариантов стирки (для разных типов белья, разной загрязненности), и человек может выбрать необходимый.

Пример 2. *Обрабатывающий станок.* Для изготовления деталей сложной формы нужно очень точно выполнить последовательность операций обработки. В случае обычных станков с непосредственным управлением соблюдение этой последовательности зависит от рабочего. При этом даже рабочий высокой квалификации может допускать ошибки, а изготовление деталей для сверхточных механизмов становится просто невозможным. Современные станки с программным управлением выполняют заранее заданную последовательность операций без участия человека. Для изготовления деталей нового вида составляется специальная программа, в которой указываются все необходимые операции.

Пример 3. *Межпланетная станция.* Предположим, нам удалось доставить на Нептун механизм, способный передвигаться по поверхности планеты и брать пробы грунта. Необходимо собрать пробы с разных участков, но при передвижении следует соблюдать осторожность, обходить возможные трещины и препятствия. Ясно, что движением этого аппарата надо как-то управлять. Сигнал от Земли до Нептуна идет 4 часа. Если человек-оператор находится на Земле, то процесс будет протекать крайне медленно: автомат передает на Землю информацию об окружающей обстановке, человек анализирует ее, принимает решение и посылает команду на Нептун. В таком режиме удастся выполнить всего несколько команд в сутки! Чтобы исследование было действительно эффективным, автомат должен самостоятельно выбирать маршрут движения в зависимости от окружающей обстановки.

Во всех рассмотренных примерах исполнители работают без вмешательства человека. Ими управляет специальное автоматическое устройство, которое определяет необходимые команды и передает их исполнителю. В роли такого устройства обычно выступает ***компьютер***. Это может быть универсальный компьютер или специализированный микропроцессор, встроенный непосредственно в устройство-исполнитель.

Откуда компьютеру известны действия, которые должен совершить исполнитель? Эту информацию он получает от человека. Человек записывает необходимую последовательность действий исполнителя и вводит ее в компьютер.

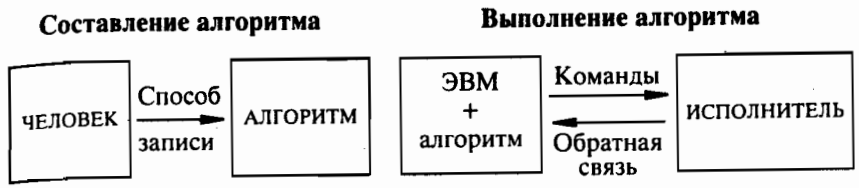


Рис. 9

Последовательность действий, записанная на специальном языке, называется ***программой***, а управление исполнителями с помощью компьютера — ***программным управлением*** (рис. 9).

Важную роль в программном управлении играют команды-вопросы. В случае, когда выбор очередной команды зависит от состояния исполнителя, необходимо включить в программу команду-вопрос и предусмотреть различные дальнейшие действия в соответствии с ответом исполнителя.

6.5. Языки программирования

Чтобы компьютер мог выполнить программу, она должна быть записана на специальном языке. Естественные языки (русский, английский и т. д.) для этого не годятся, так как в них часто встречаются многозначные слова, нередко смысл предложения зависит от того, что говорилось раньше.

Языки, на которых записываются программы для компьютера, называются ***языками программирования***. Уже сегодня существует очень много разных языков программирования, и с развитием информатики постоянно появляются все новые. Например, в 70-х гг. были популярны языки Фортран, ПЛ/1, Кобол, в 80-х и 90-х — Бейсик, Паскаль и С, а сейчас больше распространены С++ и Java.

Однако при всем различии языков программирования у них есть одно очень важное общее свойство: запись на языке программирования *однозначно* задает последовательность действий компьютера.

Мы будем использовать специальный ***школьный алгоритмический язык***. Программы на этом языке называются ***алгоритмами***, их можно выполнить на любой школьной ЭВМ, а также на любом компьютере, совместимом с IBM PC.

Одна и та же ЭВМ может понимать несколько разных языков программирования. Вот, например, как выглядит программа, которая вычисляет и выводит на экран сумму квадратов чисел от 1 до 100 на школьном алгоритмическом языке и языках программирования Бейсик и Паскаль:

Школьный алгоритмический язык

```
алг сумма квадратов
нач цел s, i
|   s:=0
|   нц для i от 1 до 100
|   | s:=s+i*i
|   кц
|   вывод "Сумма квадратов=", s
кон
```

Бейсик

```
10 S=0
20 FOR I=1 TO 100
30 S=S+I*I
40 NEXT I
50 PRINT "Сумма квадратов=";S
60 STOP
```

Паскаль

```
program sum;
   var s,i : integer;
begin
   s:=0;
   for i:=1 to 100 do s:=s+i*i;
   writeln ('Сумма квадратов=',s)
end.
```

ЗАДАЧИ И УПРАЖНЕНИЯ

1. Фирма «Электронные приборы» выпустила автоматизированную ванну «Банный комплекс-10», управляемую с помощью 10 кнопок: «долить 1 л», «долить 2 л», ..., «долить 5 л»; «слить 1 л», «слить 2 л», ..., «слить 5 л», при нажатии на которые доливается или сливается указанное количество литров воды. Однако в результате ошибки фирмы все кнопки, кроме «долить 5 л» и «слить 3 л», не работают. Как долить в ванну 3 л воды? Сколько воды при этом пропадет впустую из-за брака фирмы?

2. Автоматическое устройство имеет две кнопки и экран. При включении устройства на экране загорается число 0. При нажатии на одну кнопку число на экране удваивается (вместо x появляется $2x$). При нажатии на другую кнопку число увеличивается на 1 (вместо x появляется $x + 1$). Как надо нажимать на кнопки, чтобы на экране появилось:
   а) число 5;
   б) число 99;
   в) число 99, если разрешается нажимать на кнопки не более 10 раз?

3. а) *Волк, коза и капуста.* На берегу реки стоит крестьянин с лодкой, а рядом с ним — волк, коза и капуста. Крестьянин должен переправиться сам и перевезти волка, козу и капусту на другой берег. Однако в лодку, кроме крестьянина, помещается либо только волк, либо только коза, либо только капуста. Оставлять же волка с козой или козу с капустой без присмотра нельзя — волк может съесть козу, а коза — капусту. Как должен вести себя крестьянин?
   б) Придумайте систему команд исполнителя *Перевозчик*. Запишите последовательность команд для решения задачи о волке, козе и капусте.

4. а) Два солдата подошли к реке, по которой на лодке катаются двое мальчиков. Как солдатам переправиться на другой берег, если лодка вмещает только одного солдата (либо двух мальчиков), а солдата и мальчика уже не вмещает?
   б) А как поступить, если солдат будет не двое, а целый взвод?
   в) Сможет ли справиться с этой задачей исполнитель *Перевозчик*, которого вы придумали, решая задачу 3? Если да, запишите последовательность команд, если нет — подумайте, как усовершенствовать этого исполнителя.

5. а) Два встречных поезда, в каждом из которых паровоз и 21 вагон, встретились на дороге с одним тупиком (рис. 10). Тупик вмещает 11 вагонов или 10 вагонов и паровоз. Как поездам разъехаться (т. е. как должны маневрировать машинисты, чтобы каждый поезд продолжил движение в своем направлении)?

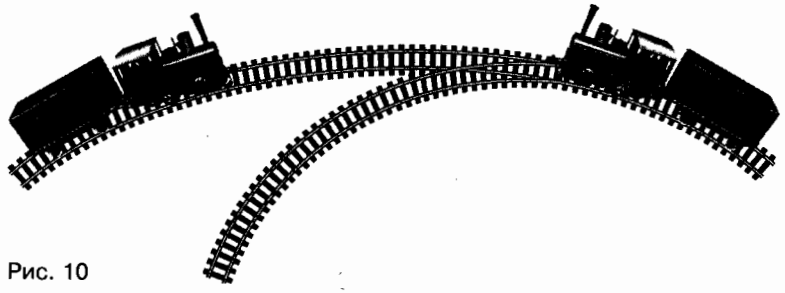


Рис. 10

   б) Придумайте систему команд исполнителя *Диспетчер*. Запишите последовательность команд, которая позволит поездам разъехаться.

6. *Ханойская башня.* На подставке укреплены три стержня. На левый стержень нанизано несколько уменьшающихся колец: внизу самое большое, на нем поменьше, сверху еще меньше и т. п. (рис. 11).

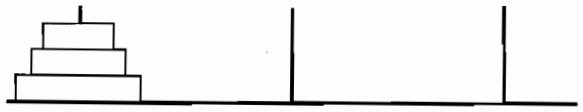


Рис. 11. Ханойская башня

Кольца можно перекладывать со стержня на стержень, соблюдая при этом два правила: 1) за один ход переносится только одно кольцо; 2) нельзя класть большее кольцо на меньшее. Необходимо перенести все кольца на правый стержень.

Опишите, как надо перекладывать кольца, если в начальный момент на левом стержне: а) 1; б) 2; в) 3; г) 4; д) 64 кольца. (По преданию, перекладыванием 64 колец занимаются монахи в одном из буддийских монастырей. В момент, когда они кончат перекладывать кольца, наступит конец света. Прикиньте, когда приблизительно это произойдет, если считать, что монахи перекладывают примерно 1 кольцо в секунду.)

Придумайте систему команд исполнителя *Монах*, занимающегося перекладкой колец. Запишите последовательность команд для решения задач а) — д).

Подсказка. Если мы умеем перекладывать n колец, то можно легко решить задачу для $n + 1$ кольца.

7. а) Есть двое песочных часов: на 3 минуты и на 8 минут. Для приготовления эликсира бессмертия его надо варить ровно 7 минут. Как это сделать, пользуясь этими часами?

б) Придумайте систему команд исполнителя *Колдун*, умеющего обращаться с песочными часами. Запишите последовательность команд этого исполнителя при изготовлении эликсира.

8. а) Петя и Коля играют в следующую игру: на стол кладется 15 спичек. Ребята по очереди берут их со стола, причем за один ход разрешается взять 1, 2 или 3 спички. Выигрывает тот, кто возьмет последнюю спичку. Первым ходит Петя. Как он должен играть, чтобы выиграть?

б) Придумайте систему команд исполнителя *Пожарник*, способного играть в эту игру. Запишите последовательность команд, следуя которой *Пожарник* будет выигрывать.

Подсказка. Не забудьте про команды-вопросы.

9. Придумайте способ нахождения самой легкой и самой тяжелой из 100 монет различной массы, если можно сделать не более 150 взвешиваний на чашечных весах без гирь.

10. Имеется а) 3; б) 4, в) 5; г) 6 монет, среди которых одна фальшивая (легче других). Придумайте способ нахождения фальшивой монеты за минимальное число взвешиваний на чашечных весах без гирь.

11. Имеется 1000 монет, из которых одна фальшивая (легче других). Придумайте способ нахождения фальшивой монеты за 7 взвешиваний на чашечных весах без гирь. Докажите, что нельзя придумать способ, который гарантирует нахождение фальшивой монеты за 6 взвешиваний.

12. Среди 11 монет не более 5 фальшивых. Все настоящие монеты имеют одинаковую массу, а про фальшивые известно только, что их масса не такая, как у настоящих. Фальшивые монеты могут быть тяжелее или легче настоящих, одинаковыми или разными. Необходимо за 10 взвешиваний на чашечных весах без гирь найти хотя бы одну настоящую монету. Хватит ли для этого 9 взвешиваний?

13. Среди $2n + 1$ различных по массе монет нужно найти среднюю (т. е. такую, которая тяжелее n монет и легче других n монет). Придумайте, как это сделать не более чем за $100n$ взвешиваний на чашечных весах без гирь.

14. Придумайте систему команд исполнителя *Эксперт*, умеющего взвешивать монеты на чашечных весах без гирь. Составьте для этого исполнителя последовательность команд для решения задач 9—13.

15. Придумайте своего исполнителя, запишите его систему команд. Составьте для него две разные задачи и решите их.

7 Алгоритмы управления исполнителями

В этом параграфе мы рассмотрим несколько простых учебных исполнителей и научимся записывать на алгоритмическом языке программы для управления ими.

7.1. Исполнитель *Стековый Калькулятор*

Исполнитель *Стековый Калькулятор* умеет выполнять четыре арифметических действия с целыми числами. У него есть память для хранения чисел (мы будем считать эту память неограниченной) и экран, на котором видны результаты вычислений.

Память *Стекового Калькулятора* устроена по принципу *стека*. Запоминаемые числа как бы кладутся одно на другое, а взять можно только то, которое лежит сверху (рис. 12).

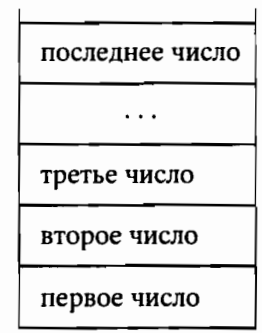


Рис. 12. Стековая память

Стеком в информатике называют любую структуру, устроенную по принципу «последним положили — первым взяли» (иногда для обозначения стека используют английское сокращение LIFO: *last in — first out*). Например, каждый стержень с кольцами из задачи о Ханойской башне (§ 6, задача 6) — это тоже стек.

Система команд *Стекового Калькулятора* описана в таблице 9, внешний вид исполнителя показан на рисунке 13.

Для выполнения любой арифметической операции необходимо, чтобы в стеке было хотя бы два числа, для команды показать необходимо одно число. Если чисел в стеке меньше, то будет отказ.

ТАБЛИЦА 9. **Система команд исполнителя *Стековый Калькулятор***

Команда	Действие исполнителя
запомнить 0	Поместить в стек число 0
запомнить 1	Поместить в стек число 1
запомнить 2	Поместить в стек число 2
запомнить 3	Поместить в стек число 3
запомнить 4	Поместить в стек число 4
запомнить 5	Поместить в стек число 5
запомнить 6	Поместить в стек число 6
запомнить 7	Поместить в стек число 7
запомнить 8	Поместить в стек число 8
запомнить 9	Поместить в стек число 9
сложить	Взять из стека два верхних числа, поместить в стек их сумму
умножить	Взять из стека два верхних числа, поместить в стек их произведение
вычесть	Взять из стека два верхних числа, поместить в стек разность второго и первого
разделить	Взять из стека два верхних числа, поместить в стек частное от деления второго на первое (остаток от деления отбрасывается)
показать	Показать на экране верхнее число в стеке

Рис. 13

Стековый Калькулятор может работать с любыми целыми числами, но помещать в стек разрешается только числа от 0 до 9. Любые другие значения могут появиться только в результате вычислений.

7.2. Пример алгоритма для *Стекового Калькулятора*

Чтобы получить на экране *Стекового Калькулятора* число 2001, можно выполнить приведенную ниже последовательность команд (нажатий на кнопки калькулятора). После каждой команды указано состояние стека.

запомнить 5	5
запомнить 5	5 5
запомнить 5	5 5 5
запомнить 4	5 5 5 4
запомнить 4	5 5 5 4 4
умножить	5 5 5 16
умножить	5 5 80
умножить	5 400
умножить	2000
запомнить 1	2000 1
сложить	2001
показать	2001

7.3. Исполнитель *Робот*

Робот действует на прямоугольном клетчатом поле. Между некоторыми клетками поля могут быть расположены стены. Какие-то клетки могут быть закрашены. Сам *Робот* всегда занимает ровно одну клетку поля (рис. 14).

Робот умеет выполнять всего 17 команд: 5 команд-приказов и 12 команд-вопросов. Мы пока изучим только команды-приказы *Робота*: вверх, вниз, вправо, влево, закрасить.

По командам вверх, вниз, вправо, влево *Робот* перемещается в соседнюю клетку в указанном направлении. Если на пути оказывается стена, команда не может быть выполнена. Например, в случае, показанном на рисунке 14, нельзя выполнить команду вверх.

По команде закрасить *Робот* закрашивает клетку, в которой стоит. Если клетка уже была закрашена, она останется закрашенной, т. е. команда будет выполнена, но никаких видимых изменений не произойдет.

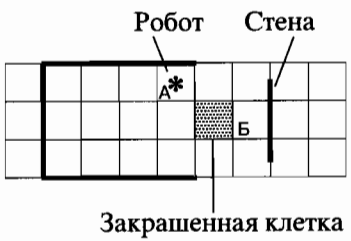


Рис. 14. Поле Робота

7.4. Непосредственное управление *Роботом*

Для непосредственного управления *Роботом* используется дистанционный пульт (рис. 15). Нажимая кнопки на пульте, человек отдает соответствующие команды, и *Робот* выполняет их. До изучения команд-вопросов мы будем пользоваться только верхней частью пульта.

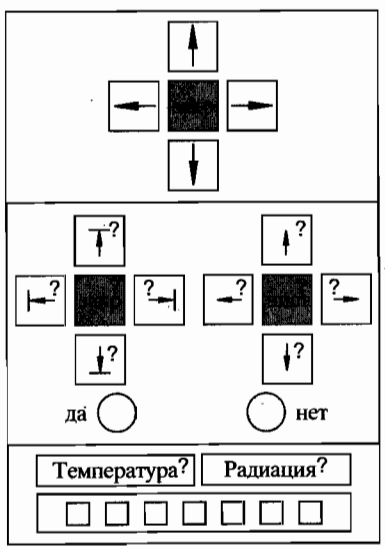


Рис. 15. Пульт Робота

Задача. В положении, изображенном на рисунке 14, с помощью непосредственного управления перевести *Робота* из клетки А в клетку Б.

■ ***Решение.*** Необходимо последовательно нажать на пульте кнопки вправо, вправо, вниз.

7.5. Программное управление исполнителем

Наша цель, однако, не ручное управление исполнителями, а составление алгоритмов для компьютера.

Для программного управления недостаточно знать, какие команды и в какой последовательности надо исполнить. Нужно еще записать эти команды в форме, понятной для компьютера, т. е. оформить их в виде алгоритма.

Например, для задачи из предыдущего пункта алгоритм будет выглядеть так:

А 1

```
алг ход конем
   дано | Робот в клетке А, стен на поле нет (рис. 14)
   надо | Робот в клетке Б (рис. 14)
нач
| вправо
| вправо
| вниз
кон
```

Стековым Калькулятором тоже можно управлять программно, причем правила оформления алгоритма будут точно такими же — ведь эти правила связаны с языком, а не с исполнителем. Например, алгоритм из пункта 7.2 можно записать так:

А 2

```
алг число 2001
   дано | стек Стекового Калькулятора пуст
   надо | на экране число 2001
нач
| запомнить 5
| запомнить 5
| запомнить 5
| запомнить 4
| запомнить 4
| умножить
| умножить
| умножить
| умножить
| запомнить 1
| сложить
| показать
кон
```

7.6. Общий вид алгоритма

В простейшем случае алгоритм на алгоритмическом языке записывается так:

```
алг имя алгоритма
   дано условия применимости алгоритма } заголовок алгоритма
   надо цель выполнения алгоритма
нач
| последовательность команд            } тело алгоритма
кон
```

Слова **алг** (алгоритм), **дано**, **надо**, **нач** (начало), **кон** (конец) называются ***служебными словами*** и предназначены для оформления алгоритма. Служебные слова **алг**, **нач** и **кон** пишутся строго одно под другим, **нач** и **кон** соединяются вертикальной чертой, правее которой помещаются команды.

Имя (название) алгоритма — это одно или несколько слов. Обычно оно подбирается так, чтобы можно было понять, для чего служит алгоритм.

В строке **дано** описывается начальное состояние, при котором должен выполняться алгоритм, в строке **надо** — состояние после выполнения алгоритма.

Строки **алг**, **дано** и **надо** образуют ***заголовок*** алгоритма. Заголовок задает условие решаемой задачи, в нем указывается, *что* делает данный алгоритм.

Часть алгоритма от строки **нач** до строки **кон** называется ***телом*** алгоритма. Тело описывает решение задачи, в нем показано, *как* достигается цель алгоритма.

7.7. Комментарии в алгоритмическом языке

В алгоритме "ход конем" (А1) после знака | в строках **дано** и **надо** записан комментарий. Такие комментарии разрешается помещать в конце любой строки, отделяя их знаком |. Если комментарий занимает несколько строк, то знак | перед комментарием надо ставить в каждой строке. Комментарии могут записываться в любой удобной для человека форме. При выполнении алгоритма компьютер полностью пропускает комментарии — алгоритм выполняется так же, как если бы комментариев вообще не было.

Таким образом, комментарии предназначены исключительно для человека — они облегчают понимание алгоритма.

7.8. Исполнение алгоритма

Получив приказ исполнить алгоритм, компьютер выполняет следующие действия:

1) Находит в памяти алгоритм с указанным именем.

2) Проверяет, соблюдаются ли начальные условия, указанные в строке **дано** (в примерах этого параграфа в **дано** пишутся только комментарии, но позднее мы научимся помещать там условия, которые компьютер сможет проверять).

3) Последовательно читает команды после строки **нач** и передает их исполнителю. Каждый такой приказ на выполнение команды называется ***вызовом*** этой команды.

4) Встретив строку **кон**, проверяет, достигнута ли цель алгоритма, указанная в строке **надо** (см. примечание к п. 2).

5) Заканчивает выполнение алгоритма.

Пример. Компьютер получает приказ исполнить алгоритм "ход конем".

1) Компьютер находит в памяти алгоритм А1.
2) В строке **дано** записан комментарий — компьютер его пропускает.
3) Компьютер последовательно командует *Роботу* вправо, вправо, вниз. *Робот* исполняет эти команды.
4) В строке **надо** помещен комментарий — компьютер его пропускает.
5) Компьютер заканчивает выполнение алгоритма "ход конем".

7.9. Ошибки в алгоритмах

Если при составлении алгоритма мы случайно вместо вниз напишем внис или вместо вправо — направо, то компьютер нашу запись не поймет и, даже не приступая к выполнению алгоритма, сообщит об ошибке. Ошибки в записи алгоритма называются ***синтаксическими***.

Но даже если все команды записаны правильно, это еще не значит, что алгоритм составлен без ошибок. Ошибки в составлении алгоритма называются ***логическими***.

Иногда логическая ошибка может привести к ***отказу*** — невозможности выполнить очередную команду. Например, при попытке выполнить алгоритм "ход конем" (А1) в обстановке, изображенной на рисунке 16, компьютер попытается последова-

тельно вызвать команды вправо, вправо, вниз. Однако вторую команду вправо *Робот* выполнить не сможет — возникает отказ. Получив от исполнителя сигнал отказа, компьютер сообщает об ошибке и прекращает выполнение алгоритма.

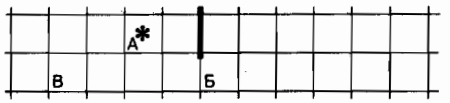


Рис. 16

У каждого исполнителя могут быть свои причины отказов. Отказ *Робота* возникает при попытке идти сквозь стену, отказ *Стекового Калькулятора* — при делении на нуль.

Ошибки в алгоритме не всегда приводят к отказам. Возможны логические ошибки, не обнаруживаемые компьютером ни до, ни во время выполнения алгоритма. Так, если в алгоритме А1 мы вместо вправо случайно напишем влево, то компьютер выполнит алгоритм, *Робот* из клетки А переместится в клетку В (см. рис. 16), но никаких сообщений об ошибках мы не получим (да и откуда компьютеру знать, куда мы на самом деле хотели переместить *Робота*!).

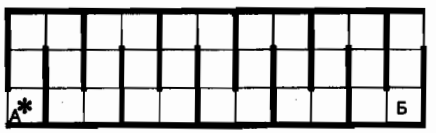
В правильно составленных алгоритмах никаких ошибок быть не должно. Но если синтаксические ошибки обычно легко устранимы, то поиск и устранение логических ошибок могут оказаться весьма трудным делом.

7.10. Запись нескольких команд в одной строке

Правила алгоритмического языка разрешают записывать в одной строке несколько команд через точку с запятой. Пусть требуется перевести *Робота* из клетки А в клетку Б (рис. 17, *а*).

Путь, который должен пройти *Робот*, можно разбить на пять одинаковых участков (рис. 17, *б*). Команды прохождения каждого участка удобно сгруппировать в одну строку — это сокращает запись алгоритма и делает его более понятным:

а)



б)

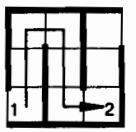


Рис. 17

алг из А в Б

дано | Робот в клетке А (рис. 17, а)

надо | Робот в клетке Б (рис. 17, а)

нач

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

кон

А 3

7.11. Исполнитель *Счетчик*

Как и *Стековый Калькулятор, Счетчик* тоже работает с целыми числами, у него тоже есть экран и память, но в памяти *Счетчика* помещается только одно число.

В системе команд *Счетчика* четыре приказа и один вопрос. Команды-приказы приведены в таблице 10, команду-вопрос мы рассмотрим несколько позже.

ТАБЛИЦА 10. **Система команд исполнителя *Счетчик***

Команды	Действие
сбросить счетчик	Число сбрасывается в нуль
увеличить счетчик	Число увеличивается на 1
уменьшить счетчик	Число уменьшается на 1
показать счетчик	Вывести число на экран

7.12. Совместное использование исполнителей

Компьютер может управлять одновременно несколькими исполнителями. Для этого достаточно включить в один алгоритм вызовы команд разных исполнителей. Например, *Робота* часто

бывает удобно использовать совместно со *Счетчиком*. В алгоритме А4 *Робот* проходит путь, показанный на рисунке 17, *б*, а *Счетчик* при этом подсчитывает количество сделанных *Роботом* шагов.

А 4

```
алг подсчет шагов
    дано | Робот в клетке 1 (рис. 17, б)
    надо | Робот в клетке 2 (рис. 17, б), на экране Счетчика количе-
         | ство сделанных Роботом шагов
нач
| сбросить счетчик
| вверх; увеличить счетчик
| вверх; увеличить счетчик
| вправо; увеличить счетчик
| вниз; увеличить счетчик
| вниз; увеличить счетчик
| вправо; увеличить счетчик
| показать счетчик
кон
```

ЗАДАЧИ И УПРАЖНЕНИЯ

1. Придумайте, как получить на экране *Стекового Калькулятора*:
а) число 1001;
б) год вашего рождения;
в) ваш почтовый индекс. Постарайтесь использовать как можно меньше команд.

2. Дан алгоритм, в котором стерты комментарии и название:

А 5

а)
```
алг
    дано |
    надо |
нач
| вверх; закрасить; вниз
| вправо; закрасить; влево
| вниз; закрасить; вверх
| влево; закрасить; вправо
кон
```


А 6

б)
```
алг
    дано |
    надо |
нач
| вверх; вправо; закрасить
| вниз; вниз; закрасить
| влево; влево; закрасить
| вверх; вверх; закрасить
| вправо; вниз
кон
```

Для каждого случая — а) и б) — опишите движение *Робота* в процессе выполнения соответствующего алгоритма. Нарисуйте начальное и конечное положения *Робота* и закрашенные в результате выполнения клетки. Придумайте подходящее название алгоритма и впишите комментарии после слов **дано** и **надо**.

3. Измените алгоритм решения задачи 2, б) так, чтобы при его исполнении *Робот*:
а) прошел тем же маршрутом, но ничего не закрашивал;
б) закрасил все клетки, в которых он побывал.

4. Известно, что на поле *Робота* нет стен и закрашенных клеток. Не делая рисунка, определите, сколько клеток будет закрашено после выполнения следующих команд:

а)	б)
закрасить	закрасить
вправо	вправо
вверх	закрасить
закрасить	закрасить
вправо	вправо
закрасить	вправо
вверх	закрасить
закрасить	закрасить
закрасить	закрасить
вправо	вправо

5. Известно, что на поле *Робота* имеется одна стена, равная по длине стороне клетки, а попытка выполнить последовательность команд из задачи 4, а) приводит к отказу. Составьте последовательность команд, при выполнении которой сохранится конечное положение *Робота*, закрашены будут те же клетки, а отказа не возникнет.

6. Составьте алгоритм, при выполнении которого *Робот* переместится из клетки А в клетку Б (рис. 18).

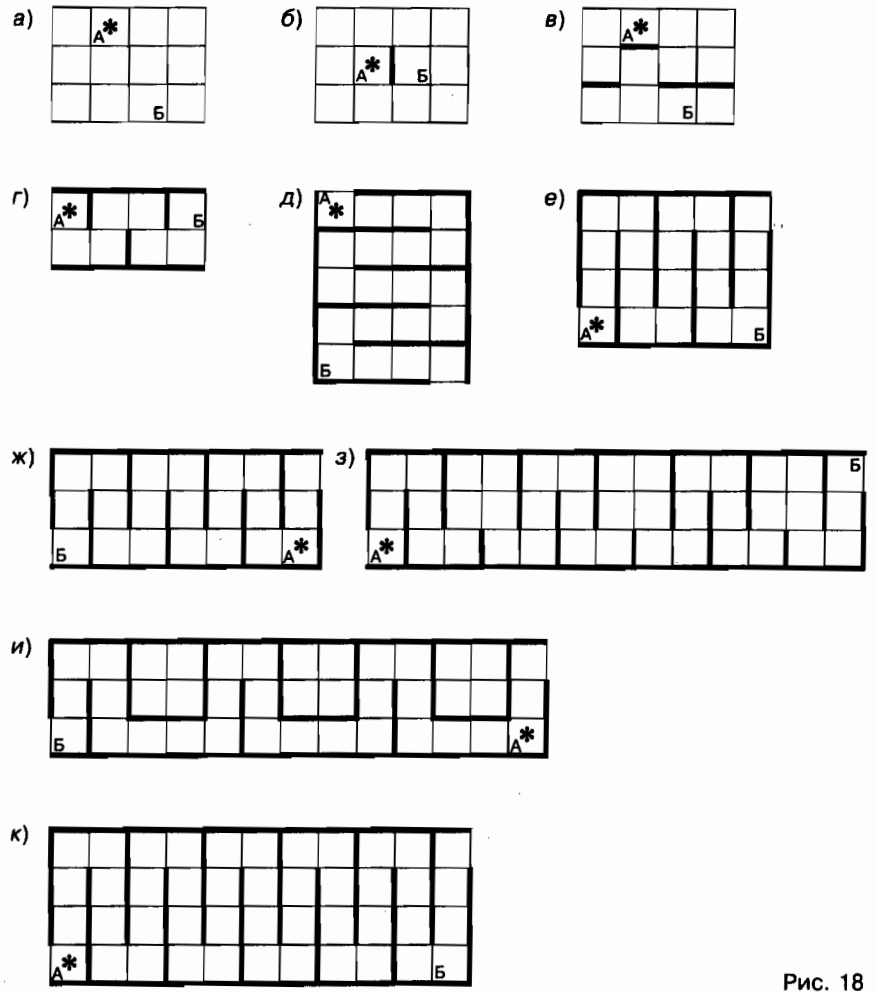


Рис. 18

7. Приведите все алгоритмы из трех команд, которые переместят *Робота* из клетки А в клетку Б в задаче 6, а).

8. Существует ли алгоритм для задачи 6, а), при выполнении которого *Робот* делает: а) два шага; б) четыре шага; в) семь шагов; г) 2000 шагов?

9. Составьте алгоритм, который переводит *Робота* из А в Б и закрашивает клетки, отмеченные точками (рис. 19).

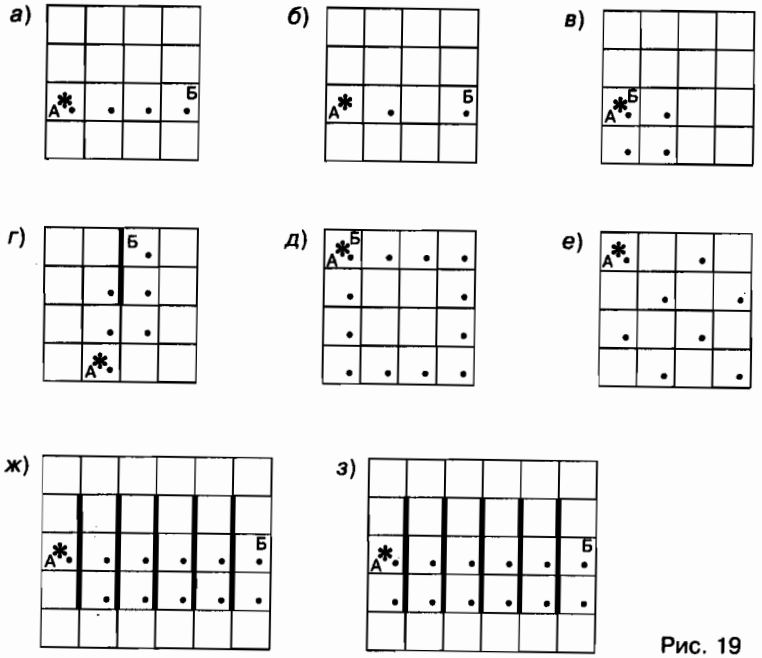


Рис. 19

10. Петя составил алгоритм, а Коля стер в нем одну команду:

А 7

алг прогулка
дано | стен на поле нет
надо | Робот погулял и вернулся в исходное положение
нач
| вверх
| вправо
| ? ? ?
| вниз
| влево
| влево
кон

Определите, какую команду стер Коля, если известно, что при выполнении составленного Петей алгоритма *Робот* возвращался в исходное положение.

11. Петя составил алгоритм, при выполнении которого *Робот* вернулся в исходное положение. Коля стер одну из команд. При выполнении Колиного алгоритма *Робот* также вернулся в исходное положение. Какую команду стер Коля?

12. Петя составил алгоритм, при выполнении которого на поле без стен *Робот* вернулся в исходное положение. Коля переставил две команды местами. Докажите, что при выполнении Колиного алгоритма *Робот* также вернется в исходное положение.

13. Петя составил алгоритм для *Робота*, который на поле без стен и закрашенных клеток закрашивает 5 клеток. Коля переставил в алгоритме две соседние команды. Может ли новый алгоритм закрашивать: а) 3; б) 4; в) 5; г) 6; д) 7 клеток?

14. Петя составил алгоритм, при выполнении которого *Робот* закрашивает 5 клеток. Коля переставил в алгоритме две команды (необязательно соседние). Может ли новый алгоритм закрашивать: а) 0; б) 1; в) 5; г) 7; д) 100 клеток?

15. Петя составил алгоритм, переводящий *Робота* из клетки А в клетку Б с закрашиванием каких-то клеток. Что должен сделать Коля с этим алгоритмом, чтобы получить алгоритм, переводящий *Робота* из Б в А и закрашивающий те же клетки?

16. Имеется некоторый алгоритм управления *Роботом*. Какие команды *Счетчика* и в какие места этого алгоритма надо добавить, чтобы *Счетчик* показал, сколько раз *Робот* закрашивал клетки? Всегда ли это число будет совпадать с количеством закрашенных клеток?

17. Имеется некоторый алгоритм управления *Роботом*. Какие команды *Счетчика* и в какие места этого алгоритма надо добавить, чтобы *Счетчик* показал, на сколько клеток правее исходного положения оказался *Робот* в конце алгоритма? Смещения *Робота* вверх и вниз не учитывать.

8 Вспомогательные алгоритмы

8.1. Понятие о вспомогательном алгоритме

Вернемся к примеру, разобранному в пункте 7.10 (с. 68). Составляя алгоритм решения задачи, мы выделили повторяющуюся часть пути *Робота*, записали ее в одну строку алгоритма и повторили эту строку необходимое число раз.

Если бы мы ошиблись при записи повторяющейся части, ошибку пришлось бы исправлять в каждой строке. Если мы захотим изменить алгоритм, чтобы приспособить его для решения другой задачи (например, пройти лабиринт на рис. 18, *к*), правку опять придется вносить в каждую строку.

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

Оформим прохождение части лабиринта, показанной на рисунке 17, *б*, в виде отдельного алгоритма.

А 8

```
алг участок
   дано | Робот в начале участка (рис. 17, б)
   надо | Робот в конце участка (рис. 17, б)
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".

А 9

```
алг из А в Б
   дано | Робот в клетке А (рис. 17, а)
   надо | Робот в клетке Б (рис. 17, а)
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. Это похоже на отца и сына: человек сам по себе не является ни отцом, ни сыном, но он может быть *чьим-то* отцом и *чьим-то* сыном.

Любой алгоритм может выполнять роль вспомогательного, для этого достаточно указать его имя в команде вызова в каком-то другом алгоритме. Здесь напрашивается аналогия с математикой: при доказательстве теорем мы часто используем уже известные теоремы, не доказывая их заново. Точно так же при составлении алгоритмов удобно 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. 20). 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 два раза обойти стену. Запишем эту мысль в виде алгоритма:

А 10

алг из А в Б с закрашиванием
дано | Робот в точке А (рис. 20)
надо | Робот в точке Б (рис. 20), нужные клетки закрашены
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):

А 11

алг закрашивание блока
дано | Робот в левом верхнем углу блока 3×5 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

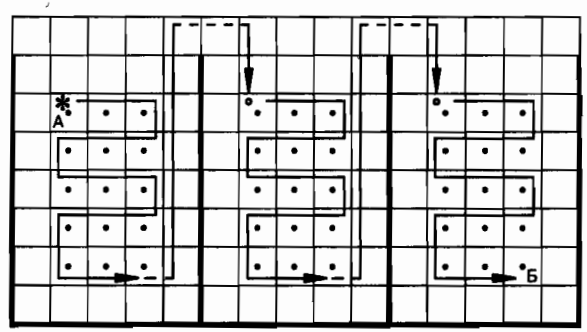


Рис. 20

алг обход стены | А 12

дано | Робот в правом нижнем углу блока (рис. 20)

надо | Робот в левом верхнем углу следующего блока (рис. 20)

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 этом для выполнения более мелких действий в свою очередь могут использоваться вспомогательные алгоритмы и т. д.

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

Такое построение алгоритмов называют также ***программированием сверху вниз***.

8.5. Заголовки вспомогательных алгоритмов

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

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

Во-вторых — 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, т. е. тело алгоритма. Здесь опять видна аналогия с математикой: чтобы пользоваться теоремой, достаточно знать ее формулировку, а не доказательство.

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.

Так, 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 таблице 11.

ТАБЛИЦА 11. **Диалог компьютера и *Робота***

Компьютер	*Робот*
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

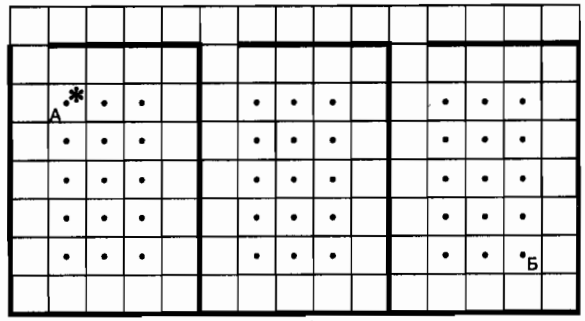
1. Измените алгоритм "закрашивание блока" так, чтобы *Робот* шел другим путем. Изменится ли при этом заголовок алгоритма? Почему?
2. Составьте алгоритм, который переводит *Робота* из А в Б и закрашивает клетки, отмеченные точками (рис. 21).
3. Составьте алгоритм, при выполнении которого в стек *Калькулятора* добавляется число 55. Используя его как вспомогательный, получите на экране число 3026.
4. Даны алгоритмы:

алг фигура — А 13
нач
| фрагмент; фрагмент; фрагмент; фрагмент; фрагмент
кон

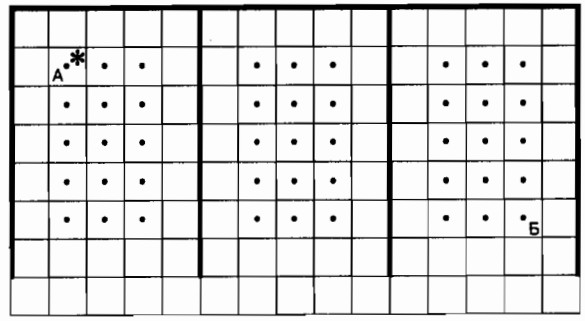
алг фрагмент — А 14
нач
| закрасить; вправо; закрасить; вправо; закрасить; вправо
| закрасить; вниз; закрасить; вниз; закрасить; вниз
| закрасить; влево; закрасить; влево; закрасить; влево
| закрасить; вверх; закрасить; вверх; закрасить
| вправо; вправо; вправо; вниз; вниз
кон

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

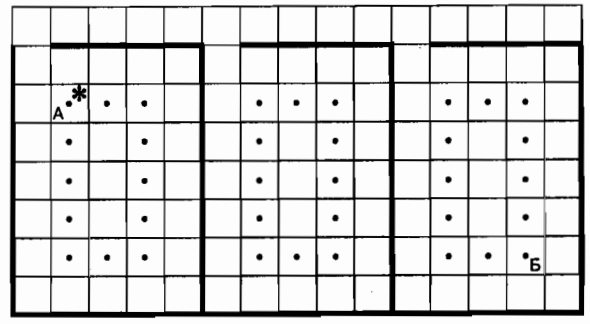
а)



б)



в)



г)

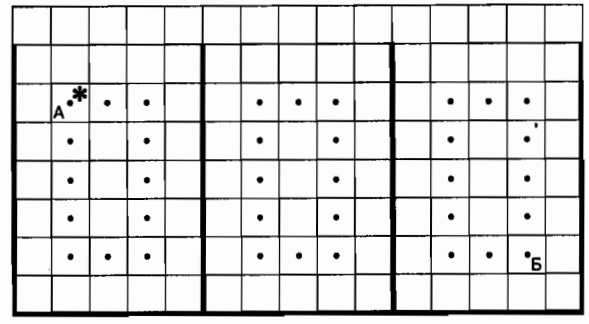
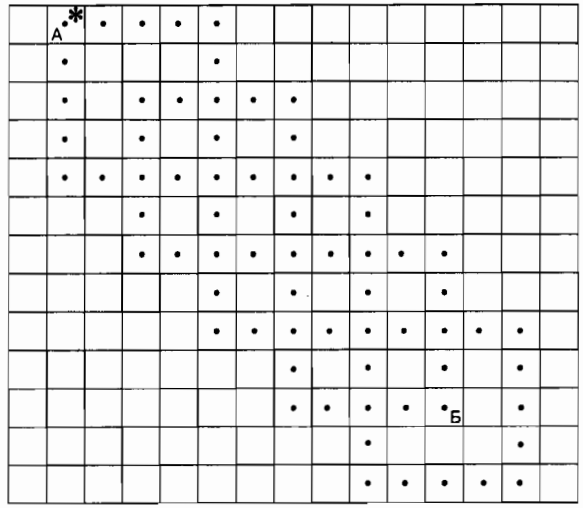
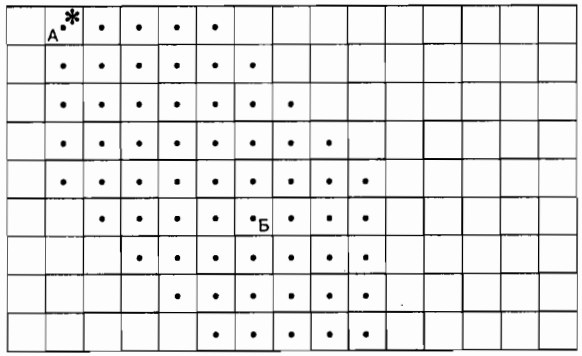


Рис. 21

а)



б)



в)

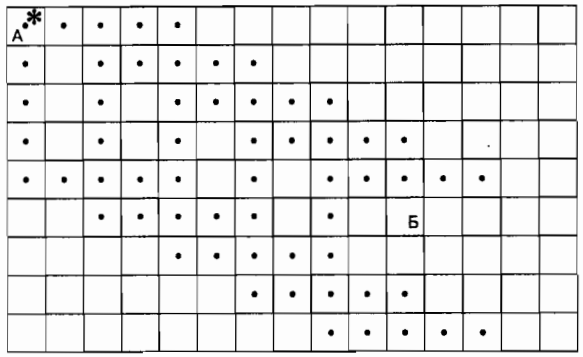


Рис. 22

5. Измените вспомогательный алгоритм "фрагмент" (А14) так, чтобы при выполнении алгоритма А13 *Робот* переместился из А в Б и закрасил клетки, отмеченные точками (рис. 22).

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, используя эти алгоритмы, переписать алгоритмы управления *Роботом*, чтобы узнать общее количество шагов, сделанных *Роботом*?

7. Петя долго решал задачу 6. Сначала он составил вспомогательные алгоритмы "вверх с подсчетом", "вниз с подсчетом", "влево с подсчетом", "вправо с подсчетом". Потом он переделал свой старый алгоритм управления *Роботом* в квадрате 9×9, вставив в начало команду "сбросить счетчик" и заменив все команды движения на "движение с подсчетом". После этого при движении *Робота* на экране счетчика все время было видно число шагов, сделанных им. Когда Петя показал свой алгоритм Коле, тот придумал новую интересную задачу:

Робот находится в левом нижнем углу квадрата 9×9 клеток. Горизонтальные и вертикальные ряды перенумерованы от 1 до 9 (рис. 23). Заменить в Петином алгоритме команды вверх с подсчетом, вниз с подсчетом *и т. д. на команды* вверх с двойным подсчетом, вниз с двойным подсчетом *и т. д. так, чтобы при движении Робота по квадрату на экране счетчика все время были видны его текущие координаты. Например, когда Робот находится в клетке Г, счетчик показывал 26, а в клетке Д — 83.*

Петя мгновенно решил эту задачу. Какие алгоритмы "вверх с двойным подсчетом", "вниз с двойным подсчетом" и т. д. составил Петя?

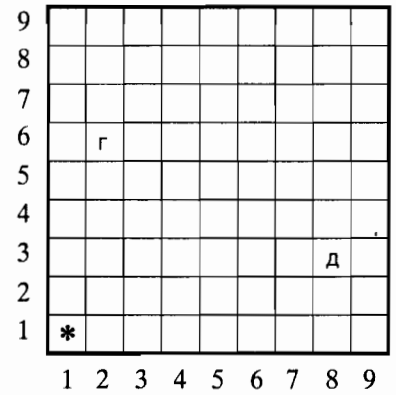


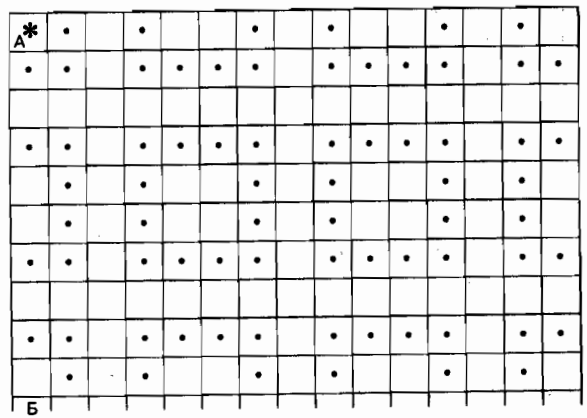
Рис. 23

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. 24).

а)



б)

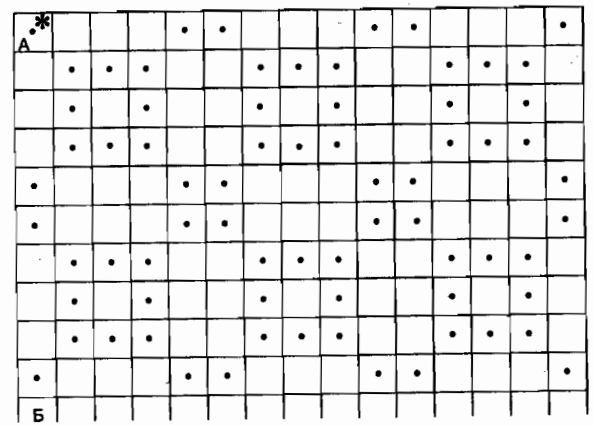


Рис. 24

11. Сколько команд закрасить даст *Роботу* ЭВМ 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**:

А 15

```
алг из А в Б
   дано | Робот в клетке А (рис. 17, а)
   надо | Робот в клетке Б (рис. 17, а)
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. 17, *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.

A 16

```
алг закрасить ряд из 10 клеток
    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

A 17

```
алг закрасить прямоугольник
    дано | на поле Робота стен нет
    надо | закрашен прямоугольник размером 10×6
         | 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 A17 можно решить и без использования вспомогательного алгоритма, вставив текст алгоритма "закрасить ряд из 10 клеток" (A16) непосредственно в текст основного алгоритма.

A 18

```
алг закрасить прямоугольник
    дано | на поле Робота стен нет
    надо | закрашен прямоугольник размером 10×6
         | 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 A18 выглядит более громоздким и менее понятным, чем A17.

9.6. Краткость и скорость не всегда совпадают

Используя цикл n **раз**, можно очень коротко записать решение некоторых задач. Вот как, например, на экране *Стекового Калькулятора* можно получить число 2001:

А 19

```
алг число 2001
    дано | стек Калькулятора пуст
    надо | на экране число 2001
нач
| запомнить 1
| нц 2000 раз
| | запомнить 1
| | сложить
| кц
| показать
кон
```

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

При выполнении алгоритма А19 компьютер выдаст *Стековому Калькулятору* 4002 команды, 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 алгоритм, на выполнение которого потребуется много времени, но для той же самой задачи может существовать другой, более 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 помощью цикла n **раз**.
3. Решение каких задач из § 8 можно упростить, воспользовавшись циклом 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 имеет возможности 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. 15). Каждому из 10 вопросов сопоставлена 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* к ней вплотную.

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 — снова скомандовать вниз, спросить снизу свободно? и т. д., пока *Робот* не ответит нет. Скомандуем ли мы при этом вниз 0, 3, 8 или 2000 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**:

А 20

```
алг вниз до стены
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. 25). Тогда при выполнении алгоритма "вниз до стены" (А20) диалог Компьютер (ЭВМ) — *Робот* будет таким:

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

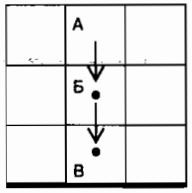


Рис. 25

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.

Графическая схема выполнения цикла **пока** изображена на рисунке 26.

При выполнении цикла компьютер повторяет следующие действия:

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

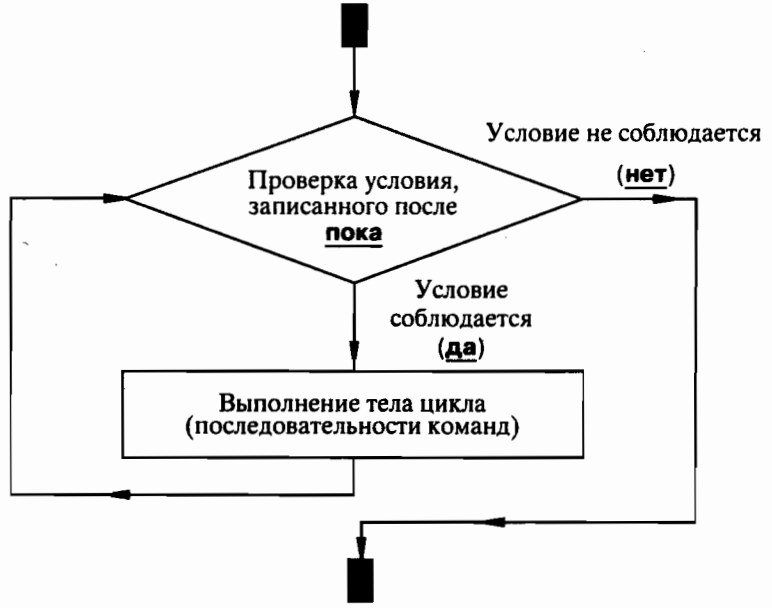


Рис. 26

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.

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 задает 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. Узнать 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, удобно 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 может служить любая команда-вопрос *Робота*. Позже мы познакомимся с другими видами простых условий.

Составное условие формулируется 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 в случае запросов к базе данных.

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. 27), 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 может оказаться любым, поэтому будет проверено и второе условие.

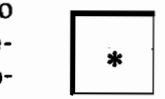


Рис. 27

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 А20 "вниз до стены" *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.

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 в процессе выполнения.

Пример 1. Пусть *Робот* находится в клетке А (рис. 28) и компьютер выполняет цикл

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

Тогда диалог Компьютер (ЭВМ) — *Робот* будет таким:

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

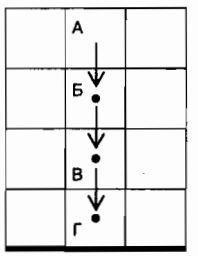


Рис. 28

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. 29) и компьютер выполняет цикл

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


Тогда диалог Компьютер (ЭВМ) — *Робот* будет таким:

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

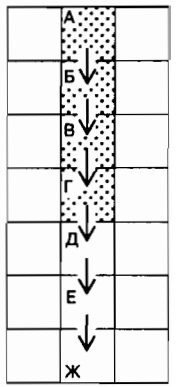


Рис. 29

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 невыполненным.

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. составить 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. 30). Внутри прямоугольника стен нет, его размеры неизвестны. Требуется закрасить горизонтальный ряд клеток от исходного положения *Робота* до правой стены и вернуть *Робота* в исходное положение.

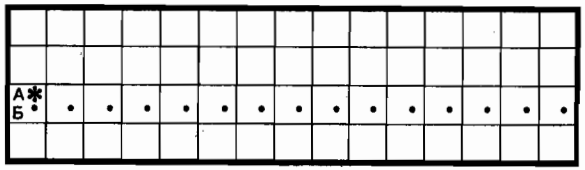


Рис. 30

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.

А 21

```
алг закрасить ряд вправо и вернуться
    дано | Робот стоит у левой стены внутри огороженного
         | с четырех сторон прямоугольника (рис. 30)
    надо | горизонтальный ряд, в левой клетке которого стоял Ро-
         | бот, полностью закрашен. Робот в исходном положении
нач
| вправо до стены с закрашиванием
| влево до стены
кон
```

Как составить вспомогательный алгоритм "влево до стены" мы уже знаем (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 исходная клетка останется незакрашенной. Поэтому перед выполнением цикла ее нужно закрасить отдельно. 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, требуется одно дополнительное закрашивание вне цикла.

Окончательно получаем алгоритм:

А 22

```
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
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. 31) и закрашивает все клетки коридора (на рис. 31 отмечены точками).

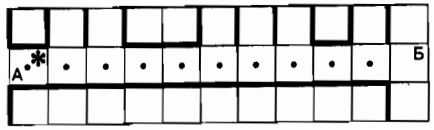


Рис. 31

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 рисунка 31, 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.

А 23

```
алг закрасить коридор
   дано | Робот в левой клетке горизонтального коридора (рис. 31)
   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. 32). Составить алгоритм, который в любом таком лабиринте перемещает *Робота* в левый верхний угол.

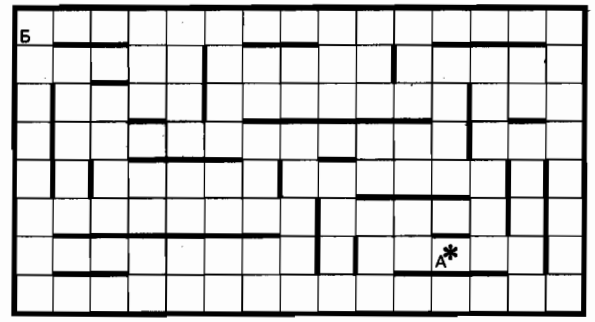


Рис. 32

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 нужно 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:

А 24

```
алг в левый верхний угол лабиринта
   дано | Робот где-то в лабиринте без углов (рис. 32)
   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 — А 25

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 А24 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 — А 26

дано | Робот в лабиринте, вид которого изображен на рис. 32

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 рисунке 33. Составьте диалог компьютера и *Робота* при выполнении цикла

нц пока сверху свободно
| вправо
кц

Рис. 33

3. Переделайте алгоритм "вправо до стены с закрашиванием" (А22), используя в нем цикл

нц пока справа свободно
| закрасить; вправо
кц

4. Используя в качестве вспомогательного алгоритм "закрасить ряд вправо и вернуться" (А21), 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. 34), Робот в исходном положении

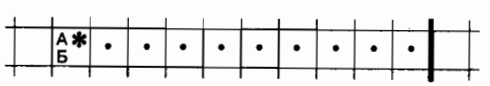


Рис. 34

б) **алг** закрасить до закрашенной клетки вправо и вернуться

дано | где-то правее Робота есть закрашенная клетка
| (рис. 35)

надо | закрашен ряд клеток между Роботом и этой клеткой,
| Робот в исходном положении (рис. 35)

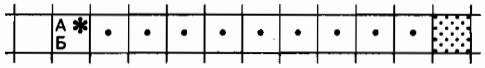


Рис. 35

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. 36)

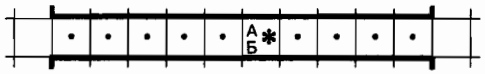


Рис. 36

г) **алг** закрасить коридор

дано | Робот где-то в горизонтальном коридоре

надо | закрашены все клетки коридора (рис. 37),
| Робот в исходном положении

Рис. 37

д) **алг** закрасить угол

дано | Робот внутри прямоугольника, огороженного
| стенами

надо | закрашены все клетки правее и выше стартовой
| (рис. 38), Робот в исходном положении

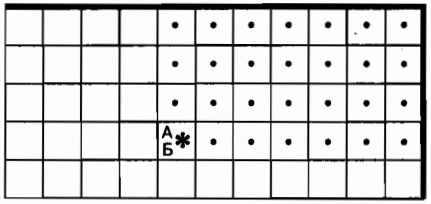


Рис. 38

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. 39). 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. 39

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. 40).

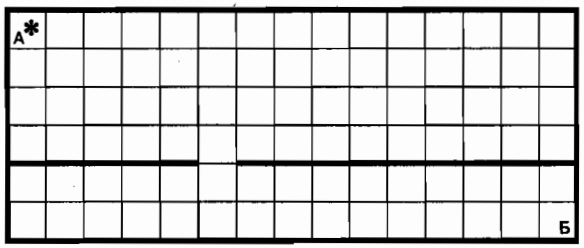


Рис. 40

12. Составьте алгоритм для закраски всех клеток вокруг прямоугольной стены (рис. 41).

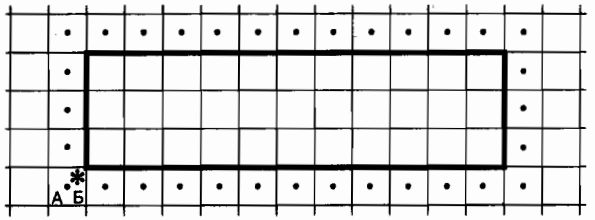


Рис. 41

13. Составьте алгоритм для закраски всех клеток вокруг Т-образной стены неизвестного размера (рис. 42).

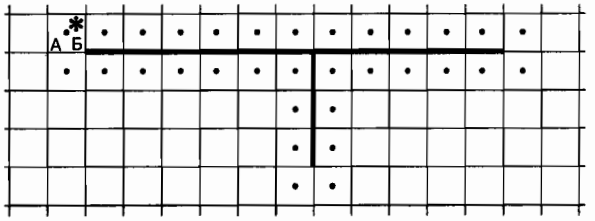


Рис. 42

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. 43). Неизвестно, в какой из двух клеток находится *Робот*. Необходимо перевести его в другую клетку.

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 клетке и должен сдвинуться влево.

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

Рис. 43

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:

А 27

```
алг дежурство
   дано | Робот в одной из клеток "двухкомнатного" помещения
        | (рис. 43)
   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. 44). Длина коридора, количество выходов и их точное расположение заранее неизвестны.

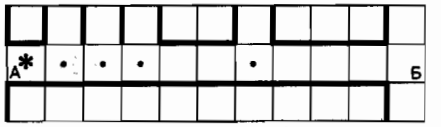


Рис. 44

Вспомним алгоритм А23 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.

А 28

```
алг разметка выходов из коридора
    дано | Робот в левой клетке горизонтального коридора (рис. 44)
    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                          если условие
| то серия 1          или             | 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** помещается 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 соблюдается, 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**.

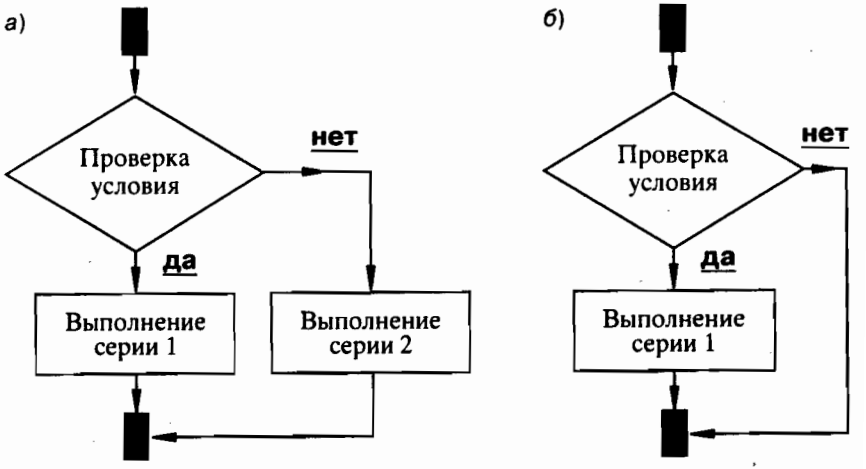


Рис. 45

Графическая схема выполнения команды **если** приведена на рисунке 45 (*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-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 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. 46).

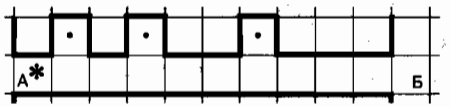


Рис. 46

Для решения этой задачи мы можем составить следующий алгоритм:

А 29

```
алг закрасить тупики
   дано | Робот стоит в левой клетке горизонтального коридора,
        | выше некоторых клеток коридора есть тупики размером
        | в одну клетку (рис. 46)
   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. 46) 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.

Задача. *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:

А 30

```
алг уйти из клетки
    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:

А 31

```
алг уйти из клетки
    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
```

Графическая схема выполнения команды **выбор** показана на рисунке 47.

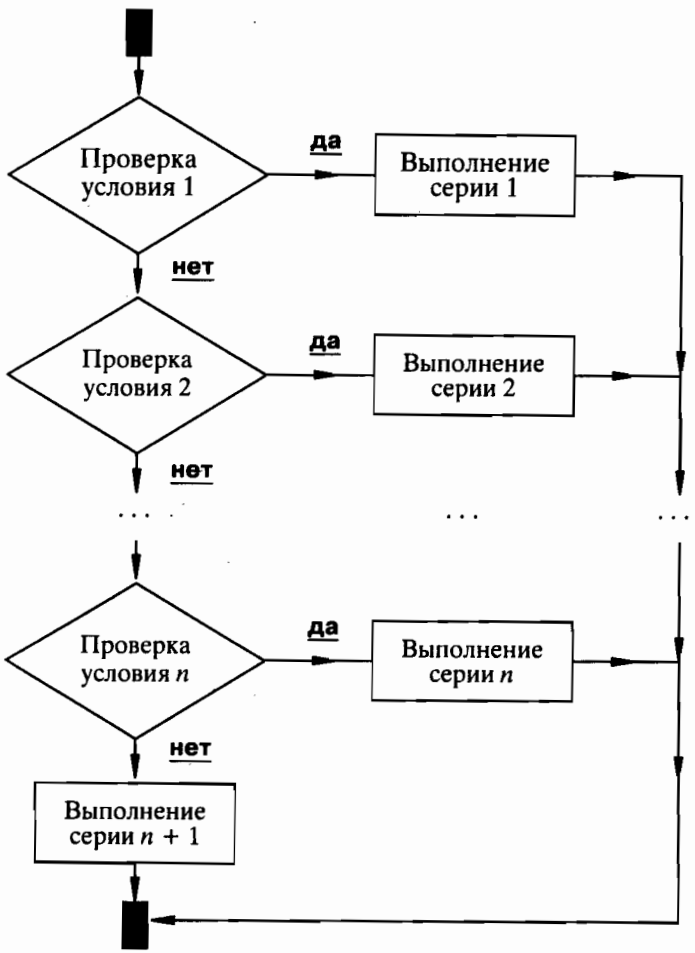


Рис. 47

Допустимо любое количество условий и соответствующих им серий команд. 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 соблюдается, то выполняется серия команд, записанная после слова **иначе**, или, если слова **иначе** нет, никакие команды не выполняются.

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 и не использовать команду **выбор**. Вот что у него получилось:

А 32

```
алг уйти из клетки
   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).

12 Анализ и тестирование алгоритмов

12.1. Результаты труда программиста

Составляя алгоритмы, мы с вами фактически были программистами. Подобную работу, только для других, намного более сложных задач, выполняют профессиональные программисты.

А что является итогом труда программиста? Предположим, в результате действия программы *Робот* выполнил какую-то полезную работу. Можно ли считать, что ее сделал программист? Нет, ее выполнил *Робот*. Может быть, программист управлял *Роботом*? Нет, этим занимался компьютер. Получается, что, хотя без программиста работа была бы невозможна, непосредственного участия в ней он вроде бы не принимает.

Рассмотрим внимательно схему программного управления (см. рис. 9 на с. 57). Видно, что единственный результат деятельности человека в этой схеме — программа работы компьютера, точнее даже не сама программа, а ***текст программы***, который компьютер обрабатывает и преобразует в исполняемые команды.

Итак, результат работы программиста — это текст написанной им программы (алгоритма).

12.2. Что такое правильный алгоритм

Если нам точно известно, в какой начальной ситуации будет выполняться тот или иной алгоритм (см., например, задачи 6 и 9 из § 7), то составить его нетрудно. Достаточно просто записать последовательность необходимых команд и оформить ее в соответствии с правилами языка. В полученном алгоритме не будет команд-вопросов — они не нужны, ведь ситуация полностью известна, следовательно, не будет в нем ни циклов **пока**, ни ветвлений. Такие алгоритмы называют ***линейными***.

Проверить правильность линейного алгоритма очень легко — достаточно выполнить одну за другой все входящие в него команды, чтобы убедиться в достижении необходимого результата или же найти ошибку.

Однако значительно чаще бывает так, что задано только общее описание начальных условий, а точная ситуация неизвестна. С такими задачами мы встречались в § 10—11. Правильно составленный алгоритм должен работать в любой ситуации, соответствующей условию.

Как проверить такой алгоритм? Конечно, для конкретной ситуации его можно исполнить и убедиться, что он работает (или не работает, тогда нужно искать ошибку). Но разных ситуаций может быть очень много, перебрать их все просто невозможно. Даже в простейшей задаче о движении *Робота* до стены (§ 10) количество ситуаций теоретически бесконечно — ведь расстояние до стены может быть любым.

Такая ситуация возникает не только с алгоритмами. Невозможно, например, перебрать все существующие треугольники, но доказав, что сумма углов треугольника всегда равна 180°, мы будем использовать это свойство для любого треугольника без дополнительной проверки. Такой прием очень распространен в математике: если какое-то свойство удается *доказать*, то потом его используют без дополнительных проверок.

Точно так же можно поступать и с алгоритмами. Если доказать, что алгоритм правильный, то нет необходимости проверять его отдельно для каждой ситуации.

А что такое правильный алгоритм? Интуитивно мы понимаем это, но для доказательства нужны четкие формулировки.

Правильность алгоритма означает, что для любой допустимой (соответствующей условию **дано**) начальной ситуации:

1) при выполнении алгоритма не может возникнуть отказ;

2) при выполнении алгоритма не может возникнуть зацикливание;

3) после завершения алгоритма будет достигнут правильный (соответствующий условию **надо**) результат.

Существуют специальные методы *доказательства правильности алгоритмов*, которые позволяют строго доказывать справедливость данных утверждений для конкретных алгоритмов. К сожалению, эти методы достаточно сложны и требуют серьезных математических знаний, выходящих за рамки школьного курса, поэтому мы вместо строгих доказательств будем проводить *рассуждения*, подтверждающие правильность алгоритмов.

12.3. Пример рассуждения, подтверждающего правильность алгоритма

A 33

```
алг вправо до стены
    дано | где-то справа от Робота на расстоянии есть стена
    надо   справа стена | Робот подошел к стене
нач
|   нц пока справа свободно
|   |   вправо
|   кц
кон
```

Докажем правильность этого алгоритма (для краткости мы будем пользоваться словом «докажем», не оговаривая каждый раз, что доказательство не является математически строгим).

1) При выполнении этого алгоритма невозможен отказ.

В самом деле, единственная команда в алгоритме, при выполнении которой возможны отказы, — это команда вправо. Она стоит первой в цикле **пока** с условием "справа свободно" и по свойству цикла выполняется только при соблюдении этого условия. Следовательно, команда вправо выполняется, только когда шаг вправо возможен, значит, отказа не произойдет.

2) При выполнении этого алгоритма невозможно зацикливание.

По условию стена справа от *Робота* есть. При каждом выполнении цикла *Робот* делает один шаг вправо, в результате расстояние до стены уменьшается на 1. Когда-нибудь это расстояние станет равным 0 и цикл, а вместе с ним и алгоритм, завершится.

3) После завершения алгоритма условие в **надо** обязательно выполняется.

Это следует из свойства цикла **пока**: после завершения цикла его условие не может быть верным.

12.4. Использование промежуточных утверждений для доказательства правильности алгоритмов

Для более сложных и длинных алгоритмов бывает трудно сразу доказать правильность алгоритма в целом. В этом случае часто

помогают промежуточные утверждения. Алгоритм получается примерно такой:

```
алг имя
    дано условие 0
    надо результат
нач
|   серия 1
|   утв условие 1
|   серия 2
|   утв условие 2
|   ...
|   серия N
|   утв условие N
|   серия N + 1
кон
```

Остается доказать, что если было выполнено условие 0, то после выполнения серии 1 будет верно условие 1, из условия 1 после выполнения серии 2 следует условие 2 и т. д.

Пример. Робот находится на прямоугольном поле без внутренних стен. Необходимо закрасить клетку в правом верхнем углу и перевести Робота в левый верхний угол.

A 34

```
алг углы
    дано | Робот на прямоугольном поле без внутренних стен
    надо сверху стена и слева стена | Робот в левом верхнем углу
         | клетка в правом верхнем углу закрашена
нач
|   вверх до стены
|   утв сверху стена
|   вправо до стены
|   утв сверху стена и справа стена | Робот в правом верхнем углу
|   закрасить
|   утв сверху стена и справа стена и клетка закрашена
|   влево до стены
кон
```

В полученном алгоритме несложно доказать правильность перехода от каждого утверждения к следующему.

12.5. Утверждения внутри цикла. Инвариант

В рассмотренном алгоритме каждое утверждение проверяется ровно один раз, так что достаточно доказать правильность фрагментов алгоритма, заключенных между утверждениями.

Иногда возникает необходимость разобраться в работе фрагмента, расположенного внутри цикла. Если при этом использовать утверждения, они будут проверяться многократно — при каждом выполнении тела цикла, поэтому для таких утверждений надо доказать, что они остаются правильными при каждом выполнении цикла.

Утверждение, расположенное в теле цикла и остающееся правильным при каждом выполнении цикла, называется ***инвариантом*** цикла. Инварианты цикла часто применяются для доказательства правильности циклических алгоритмов.

12.6. Пример алгоритма с инвариантом цикла

A 35

```
алг к правой стене с закрашиванием
    дано | где-то справа от Робота есть стена
    надо справа стена| Робот подошел к стене, все клетки от начальной положения Робота (включая его) до
                     | стены закрашены

нач
| закрасить
| нц пока справа свободно
| | утв клетка закрашена | и закрашены все клетки левее Робота
| | вправо
| | закрасить
| кц
кон
```

Докажем правильность инварианта цикла, записанного в команде **утв**. В инвариант входят и формальное утверждение, проверяемое компьютером (клетка закрашена), и дополнительный комментарий (закрашены все клетки от *Робота* и левее).

При первом выполнении цикла утверждение, очевидно, правильно. Клетка с *Роботом* закрашена, поскольку перед циклом была выполнена соответствующая команда. Левее *Робота* интересующих нас клеток нет, поэтому можно сказать, что все клетки левее *Робота* (на самом деле не существующие) закрашены.

Докажем теперь, что если при каком-то выполнении цикла утверждение верно, то оно верно и при следующем выполнении. Для этого «развернем» цикл в линейную последовательность команд, т. е. выпишем команды в том порядке, в котором они будут выполняться компьютером. Получим такой фрагмент:

```
утв клетка закрашена | закрашены все клетки от Робота и левее
   вправо
   закрасить
   | здесь заканчивается одно выполнение тела цикла
   | и начинается следующее
утв клетка закрашена | закрашены все клетки от Робота и левее
```

Предположим, что первое утверждение верно. Тогда после выполнения команды вправо закрашены все клетки левее *Робота*, а после команды закрасить закрашена и клетка, где стоит *Робот*. Следовательно, второе утверждение верно.

Таким образом, мы доказали, что инвариант цикла справедлив при первом выполнении цикла и при каждом последующем.

Отметим важное отличие инварианта от условия цикла, которое записывается после **пока**. Справедливость условия в **пока** зависит от обстановки, в которой выполняется алгоритм, предсказать правильность этого условия заранее невозможно, а после окончания цикла оно обязательно будет нарушено. Инвариант же — условие, которое обязательно будет выполнено, независимо от конкретной обстановки при выполнении. Особенно важно, что инвариант остается верным и после того, как выполнение цикла завершено.

12.7. Использование инварианта при создании алгоритмов

В предыдущем примере мы использовали инвариант для доказательства правильности уже написанного алгоритма. Иногда, особенно в сложных задачах, инвариант полезно применять на стадии написания. Сформулировав и записав в виде инварианта условие, которое должно соблюдаться при каждом выполнении цикла, можно уменьшить возможность ошибки при записи тела и условия цикла.

Пример. *Робот* находится в коридоре. Снизу от него — непрерывная горизонтальная стена, которая тянется до конца коридора. Сверху — ряд горизонтальных стен различной длины, разделенных проходами разной ширины (рис. 48). Получить на экране *Счетчика* количество горизонтальных стен над коридором. (Например, на рисунке 48 это количество равно 4.)

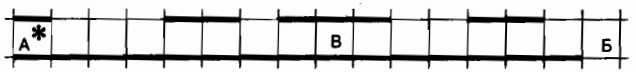


Рис. 48

Общая идея решения очевидна. *Робот* должен идти вправо по коридору, а *Счетчик* — увеличивать значение при прохождении каждой стены. Возникает вопрос: в какой момент следует учитывать очередную стену? Когда она начинается или когда заканчивается? Чтобы сделать правильный выбор, построим инвариант цикла. Если считать «начала» стен, то инвариант будет таким:

```
утв | число в Счетчике показывает количество стен, пройденных
    | Роботом, включая ту, под которой он сейчас находится
```

Если же считать «окончания» стен, инвариант получится немного другим:

```
утв | число в Счетчике показывает количество стен, полностью
    | пройденных Роботом
```

Поясним разницу этих условий. Если мы будем следовать первому из них, то в клетке В (рис. 48) значение *Счетчика* должно быть равно 3, а если второму, то — 2.

Чтобы выбрать в качестве инварианта более подходящее условие, вспомним, что инвариант должен быть верным и до и после окончания цикла. Ясно, что, когда коридор пройден до конца, количество начатых и законченных стен одинаково, т. е. соблюдены оба условия. А вот до выполнения цикла, когда *Робот* стоит в клетке А, они оказываются неравноценными. Если следовать первому условию, то до начала цикла в *Счетчике* должен быть либо 0, либо 1, в зависимости от того, есть ли стена над начальной клеткой коридора. Второе условие более определенно:

в начале в *Счетчике* обязательно должен быть нуль, так как ни одна стена еще не пройдена до конца.

Ясно, что второе условие удобнее, оно позволяет построить более простой и понятный алгоритм.

А 36

```
алг подсчет горизонтальных стен
  дано | Робот в коридоре, снизу сплошная стена, сверху
       | горизонтальные стены, разделенные проходами
  надо   снизу свободно | коридор пройден, на экране Счетчика
       | количество стен сверху от Робота
нач
| сбросить Счетчик
| нц пока снизу стена
| | утв | число в Счетчике показывает количество стен,
| |     | полностью пройденных Роботом
| | если сверху стена
| | | то | проверить, не последняя ли это клетка стены
| | |    вправо
| | |    если сверху свободно
| | |    | то | стена кончилась
| | |    |    увеличить Счетчик
| | |    все
| | | иначе вправо
| | все
| кц
| показать Счетчик
кон
```

12.8. Тестирование алгоритмов

Для реального использования на компьютере алгоритм следует проверить независимо от того, доказана ли его правильность, ведь и в доказательстве могут быть ошибки.

Чтобы проверить алгоритм, его надо исполнить и сравнить результат с тем, который требуется получить. При этом несовпадение результатов означает, что в алгоритме есть ошибки, а вот совпадение... Совпадение не значит ничего! Ведь алгоритм должен давать правильный результат для любой начальной ситуации, поэтому нельзя делать выводы после одной проверки.

Понятно, что реальных ситуаций может быть очень много, и провести проверку каждой из них невозможно. Поэтому проверяют не все, а только некоторые специально подобранные ситуации, для каждой из которых *заранее известен правильный*

результат. Такая ситуация и сама проверка называются ***тестом***, а процесс проверки — ***тестированием***.

Подбор тестов для проверки — задача трудная. Необходимо проверить не только обычные, очевидные ситуации, но и так называемые ***крайние случаи***, в которых чаще всего можно ожидать проявления ошибок в алгоритме. Понятие крайнего случая поясним на нескольких примерах.

Пример 1. Где-то справа от *Робота* есть стена. Крайний случай — *Робот* уже у стены.

Пример 2. *Робот* где-то в коридоре. Крайние случаи — *Робот* в крайней левой клетке коридора; *Робот* в крайней правой клетке коридора; *Робот* в коридоре из одной клетки.

Пример 3. Некоторые клетки коридора закрашены. Крайние случаи — закрашенных клеток нет; закрашена ровно одна клетка; закрашены все клетки коридора.

Задача 1. Дан фрагмент алгоритма:

```
нц пока справа свободно
| вправо
| если клетка закрашена
| | то влево
| | иначе вверх
| все
кц
```

Придумайте ситуации, в которых при выполнении этого фрагмента:

а) произойдет отказ;

б) произойдет зацикливание;

в) компьютер не даст *Роботу* ни одной команды-приказа;

г) компьютер даст *Роботу* ровно одну команду-приказ;

д) компьютер даст *Роботу* ровно две команды-приказа.

■ ***Решение.*** а) Из всех команд данного фрагмента отказ возможен только при выполнении команд движения Робота. Этих команд во фрагменте три.

Команда вправо выполняется только при соблюдении условия цикла "справа свободно", значит, при ее выполнении отказ невозможен.

Команда влево выполняется сразу после команды вправо. Если *Робот* смог пройти вправо, он сможет и вернуться обратно. Значит, и здесь отказ невозможен.

А вот команда вверх никак не защищена, при ее выполнении отказ действительно может случиться. Это произойдет, если, шагнув вправо, *Робот* попадет на чистую клетку, выше которой расположена стена (рис. 49, *а*).

б) Одна из возможных причин зацикливания — отсутствие изменений в обстановке при выполнении цикла. В данном фрагменте такое возможно, если *Робот* будет делать шаг вправо, а затем, попав на закрашенную клетку, возвращаться влево (рис. 49, *б*).

в) *Робот* не получит команд-приказов, если условие цикла с самого начала не будет выполнено, т. е. в начальном положении справа от *Робота* будет находиться стена (рис. 49, *в*).

г) Если условие цикла с самого начала не выполнено, *Робот* вообще не получит приказов. Если условие выполнено, *Робот* получит приказ вправо, а затем, в зависимости от того, закрашена ли клетка, в которую он попал, приказ вверх или вниз. Таким образом, ситуация, в которой *Робот* получит ровно один приказ, для данного фрагмента невозможна.

д) Приказов будет ровно два, если цикл будет выполнен один раз. Соответствующая ситуация изображена на рисунке 49, *г*.

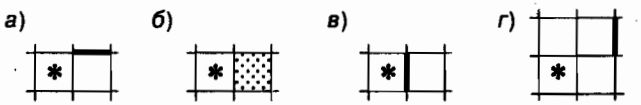


Рис. 49

Задача 2. Что можно сказать о правильности такого фрагмента:

```
нц пока клетка закрашена
| если справа свободно
| | то вправо; закрасить
| все
кц
```

■ ***Решение.*** На первый взгляд задание бессмысленно: что можно сказать о правильности фрагмента, если неизвестно, какую задачу он должен решать?

Действительно, если цель неизвестна, невозможно утверждать, что действия верны. Однако бывают ситуации, когда ошибку можно увидеть, даже не зная, для чего составлялся алгоритм.

Посмотрим внимательно на приведенный фрагмент. Если тело цикла будет исполнено хотя бы один раз, то обязательно произойдет зацикливание! В самом деле, если справа от *Робота* свободно, то он перейдет на новую клетку и закрасит ее, после чего условие цикла "клетка закрашена" будет заведомо верным. Если же справа от *Робота* стена, он останется в той же клетке.

Итак, данный цикл либо не будет выполнен ни разу, либо приведет к зацикливанию. Понятно, что в правильных алгоритмах такая ситуация встречаться не должна, так что фрагмент, вероятно, содержит ошибку.

ЗАДАЧИ И УПРАЖНЕНИЯ

1. Дан фрагмент алгоритма:

```
если клетка чистая
| то
|   нц пока справа свободно
|   | закрасить; вправо
|   кц
|   иначе влево
все
```

Придумайте ситуации, в которых при выполнении этого алгоритма:
а) произойдет отказ;
б) произойдет зацикливание;
в) *Робот* не получит ни одного приказа;
г) *Робот* получит ровно два вопроса;
д) *Робот* получит нечетное число приказов.

2. Составьте набор тестов для проверки решения задач 7 и 9 из § 10.

3. Составьте набор тестов для проверки решения задач 4—6 из § 11.

4. *Робот* находится внутри прямоугольника, с четырех сторон огороженного стенами. Внутри прямоугольника стен нет. В левой вертикали закрашены некоторые клетки. Закрасить соответствующие им клетки в правой вертикали.

5. *Робот* находится внутри прямоугольника, с четырех сторон огороженного стенами. Внутри прямоугольника стен нет. Некоторые клетки в прямоугольнике закрашены. Закрасить горизонтальные ряды, в которых:
а) закрашена левая клетка;
б) закрашены левая и правая клетки;
в) закрашены левая и правая клетки и еще хотя бы одна;

г) закрашены левая и правая клетки и еще ровно одна;
д) есть хотя бы одна закрашенная клетка;
е) есть ровно одна закрашенная клетка;
ж) есть хотя бы две закрашенные клетки;
з) есть ровно две закрашенные клетки;
и) четное число закрашенных клеток;
к) нечетное число закрашенных клеток.

6. *Робот* находится внутри прямоугольника, с четырех сторон огороженного стенами. Внутри прямоугольника стен нет. Одна клетка в прямоугольнике закрашена.
а) Составить алгоритм, приводящий *Робота* в закрашенную клетку.
б) Закрасить горизонталь и вертикаль, на пересечении которых расположена эта клетка.

7. *Робот* находится внутри прямоугольника, с четырех сторон огороженного стенами. Внутри прямоугольника нет стен и закрашенных клеток. Закрасить часть клеток так, чтобы оставшиеся образовали квадрат максимально возможных размеров (рис. 50).

8. *Робот* стоит на поле без стен.
а) На поле две закрашенные клетки — одна где-то правее *Робота*, другая — где-то выше *Робота*. Закрасить прямоугольник с вершинами в этих клетках (рис. 51).
б) Где-то правее *Робота* есть закрашенная клетка. Закрасить квадрат с вершинами в этой клетке и в клетке начального положения *Робота* (рис. 52).

9. *Робот* находится внутри прямоугольника, с четырех сторон огороженного стенами. Внутри прямоугольника стен нет. Закрасить клетки прямоугольника в шахматном порядке.

10. *Робот* помещен внутрь прямоугольника, с четырех сторон огороженного стенами. Внутри прямоугольника стен нет, некоторые клетки закрашены. Закрасить горизонтали и вертикали всех закрашенных клеток.

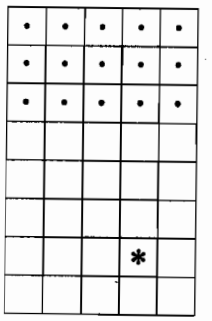


Рис. 50

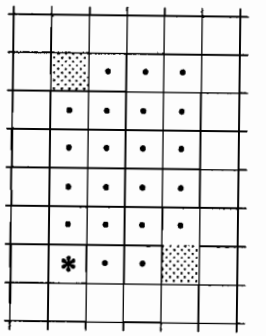


Рис. 51

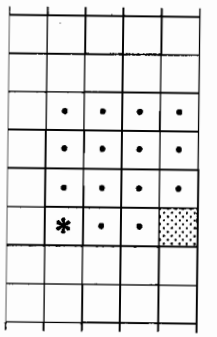
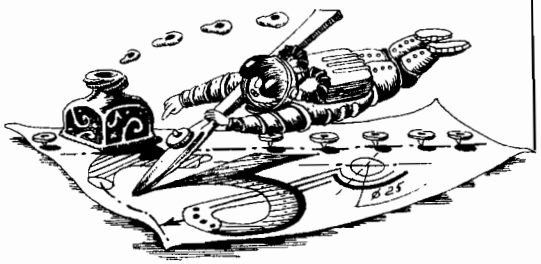


Рис. 52

ГЛАВА 3

Алгоритмы и величины



13 Исполнитель *Чертежник* и работа с ним

13.1. Особенности записи чисел в информатике

В информатике для отделения целой части числа от дробной используется точка, а не запятая, как в школьной математике (например, 1.7 или 2.5). Это позволяет записывать несколько рядом стоящих чисел через запятую без риска вызвать путаницу (например, 1.7, 2.5, −11.3, 2, 3.14). При задании точек плоскости координаты x и y в информатике разделяются запятой (рис. 53).

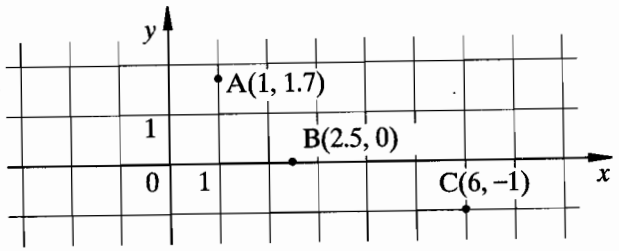


Рис. 53

13.2. Исполнитель *Чертежник*

Чертежник предназначен для построения рисунков, чертежей, графиков и т. д. на бесконечном листе бумаги. *Чертежник* имеет перо, которое можно поднимать, опускать и перемещать. При перемещении опущенного пера за ним остается след — отрезок от старого положения пера до нового. Всего *Чертежник* умеет выполнять четыре команды:

опустить перо
поднять перо
сместиться в точку (**арг вещ** x, y)
сместиться на вектор (**арг вещ** a, b)

По команде опустить перо *Чертежник* опускает перо. Если перо уже было опущено, *Чертежник* не выполняет никаких действий, но и отказа не происходит. Таким образом, после выполнения команды опустить перо перо оказывается опущенным (готовым к рисованию) независимо от его предыдущего положения.

Аналогично по команде поднять перо перо будет поднято. Выполнение этой команды тоже не может привести к отказу.

Команды сместиться в точку и сместиться на вектор перемещают перо *Чертежника*. Если при этом перо опущено, на бумаге остается след. Таким образом, эти команды позволяют строить чертежи и рисунки.

13.3. Команды с параметрами

В отличие от *Робота*, который всегда двигался ровно на одну клетку, смещение *Чертежника* может быть произвольным. Поэтому для выполнения команд сместиться в точку и сместиться на вектор необходимо задать дополнительную информацию — указать, куда надо переместить перо *Чертежника*.

Эта дополнительная информация записывается в команде в виде ***аргументов*** — *вещественных* (действительных) чисел, которые помещаются в скобках после имени команды. Например: сместиться в точку (2, 3) или сместиться на вектор (1.4, 2.3).

Необходимость аргументов указывается в описании команды:

сместиться в точку (**арг вещ** x, y)
сместиться на вектор (**арг вещ** a, b)

Подробнее смысл слов **арг** и **вещ** раскрывается в § 14.

13.4. Абсолютное и относительное смещение

В команде сместиться в точку в качестве аргументов указываются координаты той точки, в которую попадет перо после выполнения команды. На рисунке 54, *а* показаны результаты выполнения команды сместиться в точку (2, 3) при различных положениях пера до этой команды. Видно, что независимо от предыдущего положения перо оказывается в точке (2, 3), но длина и направление отрезка, который при этом чертится (конечно, если перо опущено), могут быть различны. Команду сместиться в точку называют командой ***абсолютного смещения***, так как в ней указываются ***абсолютные координаты*** пера.

Несколько иначе работает команда сместиться на вектор. Если перо *Чертежника* находится в точке (x, y), то по команде сместиться на вектор (a, b) *Чертежник* передвинет перо в точку с координатами $(x + a; y + b)$. Таким образом, координаты, указанные в команде, измеряются не от начала координат, а *относительно* текущего положения пера *Чертежника*. Поэтому команду сместиться на вектор называют командой ***относительного смещения***.

На рисунке 54, *б* показаны результаты выполнения команды сместиться на вектор (2, 3) при различных положениях пера до этой команды. Из рисунка видно, что положение пера после этой команды зависит от его предыдущего положения, но зато в результате получается отрезок, длина и направление которого постоянны. В математике такой отрезок называется ***вектором***, отсюда и происходит название команды.

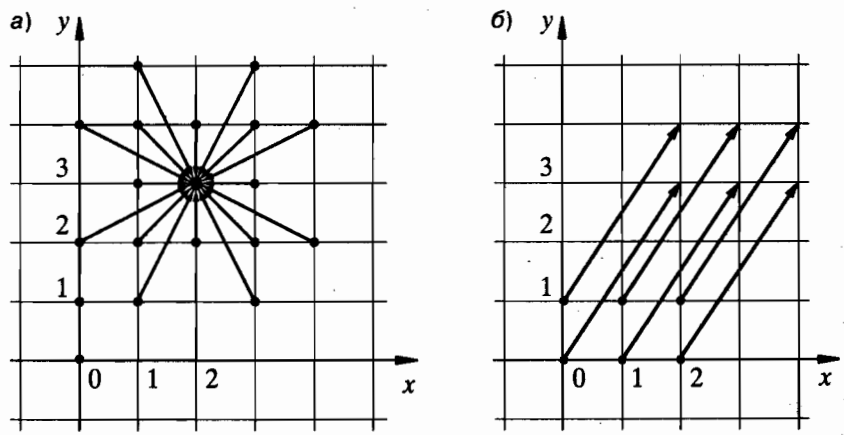


Рис. 54

Команды абсолютного смещения создают рисунок в строго определенном месте координатной плоскости. Они обычно используются, когда рисунок привязан к месту, например при построении графиков функций.

Команды относительного смещения позволяют создавать рисунок в любом месте. Они применяются для создания рисунков, у которых точное место не играет роли или которые нужно воспроизводить в разных местах.

13.5. Пример алгоритма управления *Чертежником*

Попробуем с помощью *Чертежника* изобразить прямоугольник с двумя диагоналями (рис. 55, *а*). Как это сделать? Можно начать с левого нижнего угла и, двигаясь против часовой стрелки, нарисовать все четыре стороны прямоугольника, не отрывая пера от бумаги, после чего провести диагонали (рис. 55, *б*).

Какие же команды и в какой последовательности нужно для этого дать *Чертежнику*? Пусть в начальный момент перо поднято. В левый нижний угол можно попасть по команде

сместиться в точку (1, 1)

Теперь надо нарисовать прямоугольник. Для этого прежде всего следует вызвать команду

опустить перо

После этого перо будет расположено в точке (1, 1) и опущено. Для рисования прямоугольника воспользуемся командой сместиться в точку:

сместиться в точку (3, 1)
сместиться в точку (3, 2)
сместиться в точку (1, 2)
сместиться в точку (1, 1)

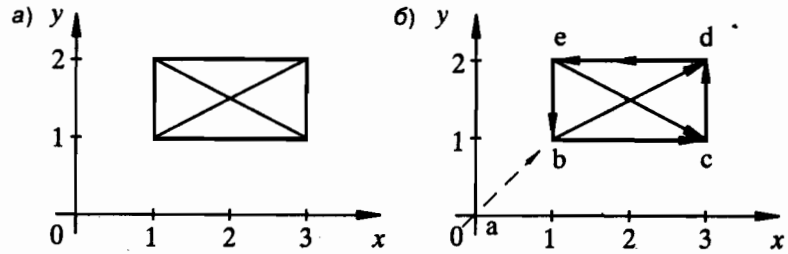


Рис. 55

После этого прямоугольник будет нарисован, а опущенное перо окажется снова в левом нижнем углу — в точке (1, 1).

Осталось нарисовать диагонали. Это можно сделать так:

сместиться в точку (3, 2)
сместиться в точку (1, 2)
сместиться в точку (3, 1)

При этом, однако, верхняя сторона прямоугольника будет нарисована второй раз. Если мы хотим этого избежать, то перед командой

сместиться в точку (1, 2)

нужно поднять перо, а потом его опустить:

сместиться в точку (3, 2)
поднять перо
сместиться в точку (1, 2)
опустить перо
сместиться в точку (3, 1)

В этот момент вся картинка изображена и осталось поднять перо, чтобы лист бумаги можно было вынуть:

поднять перо

Запишем полученный алгоритм на алгоритмическом языке:

алг прямоугольник с диагоналями — А 37
дано | перо поднято
надо | нарисован прямоугольник с диагоналями (рис. 55, *а*), перо поднято

```
нач
| сместиться в точку (1, 1)
| опустить перо
| сместиться в точку (3, 1)
| сместиться в точку (3, 2)
| сместиться в точку (1, 2)
| сместиться в точку (1, 1)
| утв | нарисован прямоугольник
| сместиться в точку (3, 2)
| поднять перо
| сместиться в точку (1, 2)
| опустить перо
| сместиться в точку (3, 1)
| поднять перо
кон
```


13.6. Рисование букв

С помощью *Чертежника* можно рисовать любые фигуры, построенные из отрезков, например буквы. Составим алгоритм, при выполнении которого *Чертежник* рисует на клетчатой бумаге букву М (рис. 56; размеры каждой клетки 1×1). Поскольку начальное положение пера на плоскости не задано, то придется воспользоваться командой сместиться на вектор:

алг буква М — А 38
дано | перо в точке А (рис. 56) и поднято
надо | нарисована буква М (рис. 56), перо в точке Б и поднято

```
нач
| опустить перо
| сместиться на вектор (0, 4)
| сместиться на вектор (1, –2)
| сместиться на вектор (1, 2)
| сместиться на вектор (0, –4)
| поднять перо
| сместиться на вектор (1, 0)
кон
```

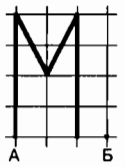


Рис. 56

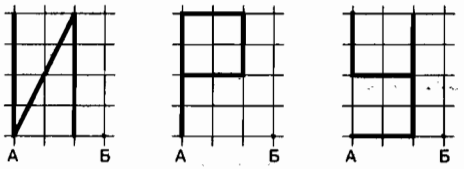


Рис. 57

Аналогично можно составить алгоритмы "буква И", "буква Р" и "буква У" (рис. 57).

13.7. Использование вспомогательных алгоритмов

Записав алгоритмы для рисования букв, можно использовать их как вспомогательные и составлять алгоритмы рисования слов.

алг МИР — А 39
дано | перо поднято
надо | нарисовано слово МИР, перо поднято и расположено в конце слова (в начале следующей буквы)

```
нач
| буква М
| буква И
| буква Р
кон
```

Подобным образом можно строить с помощью *Чертежника* любые сложные изображения, используя вспомогательные алгоритмы для рисования стандартных деталей.

Разбивая изображение на части и составляя отдельные алгоритмы для каждой из них, важно точно учитывать положение пера до и после рисования каждого фрагмента. Например, при построении букв мы соблюдали следующее соглашение: перед началом рисования перо находится в левом нижнем углу буквы, после окончания — в левом нижнем углу следующей буквы слова. Подобные соглашения рекомендуется принимать при разработке любого проекта. Лучше всего заносить их в условия **дано** и **надо**.

ЗАДАЧИ И УПРАЖНЕНИЯ

1. Петя записал через запятую несколько вещественных и целых чисел, по привычке поставив десятичные запятые внутри чисел. Вот что у него получилось:
   а) 3,5,7; б) 7,3,5,0,1.
   Сколькими способами можно прочесть эти записи, если в а) записано два числа, а количество чисел, записанных в б), неизвестно?

2. Нарисуйте результат выполнения следующего алгоритма:

A 40

```
алг домик
    дано | перо поднято
    надо | нарисован домик, перо в исходном положении
         | и поднято
нач
| опустить перо
| сместиться на вектор (4, 0)
| сместиться на вектор (0, 4)
| сместиться на вектор (–4, 0)
| сместиться на вектор (0, –4)
| поднять перо
| сместиться на вектор (0, 4)
| опустить перо
| сместиться на вектор (2, 2)
| сместиться на вектор (2, –2)
| поднять перо
| сместиться на вектор (–4, –4)
кон
```

3. Измените алгоритм "домик" (А40) так, чтобы домик рисовался с окошком.

4. Дан основной алгоритм "улица из трех домиков":

A 41

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

Этот алгоритм использует вспомогательный алгоритм "домик" (А40). Нарисуйте результат выполнения алгоритма А41 (полученную картинку и положение пера *Чертежника*).

5. Составьте алгоритм рисования улицы из шести домиков.

6. Петя зачеркнул последнюю команду сместиться на вектор (–4, –4) в алгоритме "домик" (А40). Как Коля должен изменить алгоритм "улица из трех домиков" (А41), чтобы рисовалась та же картинка, что и раньше?

7. Составьте алгоритм управления *Чертежником*, после выполнения которого будут нарисованы:
   а) отрезок с концами в точках (1, 2) и (–1, 1);
   б) квадрат со сторонами длины 4, параллельными координатным осям, так, чтобы левый нижний угол квадрата совпадал с начальным положением пера *Чертежника*;
   в) квадрат со сторонами длины 6, параллельными координатным осям, так, чтобы левый нижний угол квадрата совпадал с начальным положением пера *Чертежника*;
   г) какой-нибудь отрезок длины 3, проходящий через точку (2, 2);
   д) какой-нибудь квадрат со сторонами длины 2 и центром в начале координат;
   е) какой-нибудь прямоугольник с длинами сторон 3 и 4, содержащий внутри себя начало координат;
   ж) какой-нибудь параллелограмм.

8. Составьте алгоритм управления *Чертежником*, после исполнения которого будут нарисованы:
   а) инициалы полководца Кутузова;
   б) ваши инициалы;
   в) буква «Ф»;
   г) зеркальные отражения букв «И» и «Р» относительно горизонтальной оси;
   д) число 12 римскими цифрами;
   е) слово «МГУ»;
   ж) почтовый индекс 161110 (цифры индекса должны быть написаны как на почтовых конвертах).

9. Составьте алгоритм для рисования фигур, изображенных на рисунке 58, так, чтобы в процессе рисования перо не отрывалось от бумаги и ни одна линия не проводилась дважды.

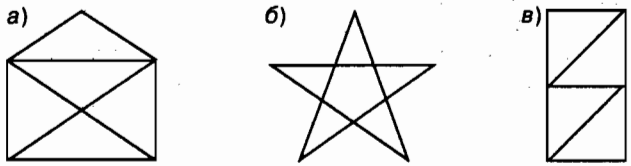


Рис. 58

10. Составьте десять алгоритмов для изображения десяти цифр почтового индекса так, чтобы при их последовательном вызове цифры рисовались друг за другом. Используя эти алгоритмы как вспомогательные, напишите ваш почтовый индекс.

11. По образцу алгоритма "МИР" (А39) составьте алгоритмы: а) РИМ; б) МИМ.

12. Измените алгоритмы рисования букв «М», «И», «Р» так, чтобы при последовательном вызове этих алгоритмов слово «МИР» оказалось написанным:
а) с удвоенным расстоянием между буквами;
б) буквами удвоенного размера;
в) сверху вниз;
г) сверху вниз буквами удвоенного размера.

13. Дан алгоритм:

А 42

алг фигура
дано | перо в начале координат и поднято
нач
| сместиться в точку (2, 1)
| опустить перо
| сместиться на вектор (0, 3)
| сместиться на вектор (1, 0)
| сместиться на вектор (0, –1)
| сместиться на вектор (1, 0)
| сместиться на вектор (0, –1)
| сместиться на вектор (1, 0)
| сместиться на вектор (0, –1)
| сместиться в точку (2, 1)
| поднять перо
кон

а) не выполняя алгоритма и не рисуя получившейся фигуры, определите, где будет расположено перо после выполнения алгоритма, будет ли оно поднято или опущено;

б) выполните алгоритм, нарисуйте получившуюся фигуру;
в) переделайте алгоритм так, чтобы он рисовал где-нибудь на плоскости фигуру вдвое большего размера;
г) переделайте алгоритм так, чтобы он рисовал фигуру, симметричную первой относительно оси *y*;
д) определите, что будет нарисовано, если в алгоритме изменить знаки всех аргументов на противоположные.

14. Дан алгоритм:

А 43

алг ломаная
дано | перо в начале координат и поднято
нач
| опустить перо
| сместиться на вектор (1, 3)
| сместиться на вектор (1, 2)
| сместиться на вектор (1, 1)
| сместиться на вектор (1, 0)
| сместиться на вектор (1, –1)
| сместиться на вектор (1, –2)
| сместиться на вектор (1, –3)
| поднять перо
кон

Не выполняя алгоритма и не рисуя получившейся ломаной, определите:
а) будет перо после выполнения поднято или опущено;
б) координату *x* конечного положения пера;
в) координату *y* конечного положения пера;
г) будет ли ломаная замкнута;
д) расстояние между концами ломаной.
Нарисуйте ломаную, проверьте ваши ответы.

15. Составьте алгоритм управления *Чертежником*, после исполнения которого будет нарисован квадрат 4×4, заштрихованный горизонтальными и (или) вертикальными линиями на рисунке 59 (расстояние между линиями равно 0,4).

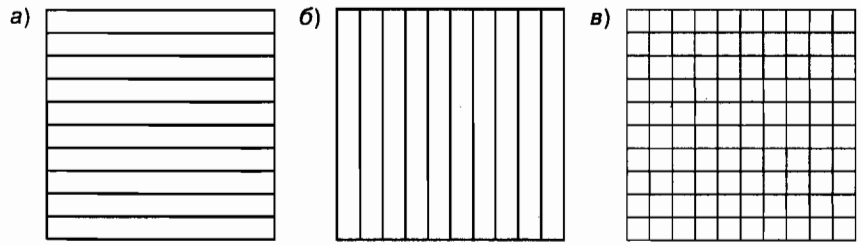


Рис. 59

16. Вспомогательный алгоритм "картинка" рисует некоторую картинку в квадрате 1×1 и возвращает перо в начальное положение — левый нижний угол квадрата. Составьте алгоритм, рисующий 100 экземпляров картинки в квадрате 10×10.

14 Алгоритмы с аргументами

14.1. Пример алгоритма с аргументом

Вспомните упражнения 7, б), в) из предыдущего параграфа. Для рисования квадратов с длинами сторон 4 и 6 вы составляли два разных алгоритма, отличающихся только *числами* в командах.

А как быть, если нужно рисовать много разных квадратов с разными длинами сторон? Можно, конечно, составить множество похожих алгоритмов, но делать этого очень не хочется! Было бы намного удобнее создать единый алгоритм-образец, в который компьютер сам подставлял бы каждый раз нужные числа.

Для подобных целей в алгоритмическом языке существуют алгоритмы с ***аргументами***. Например, алгоритм рисования квадрата с произвольной длиной стороны может выглядеть так:

А 44

```
алг квадрат (арг вещ а)
    дано | перо Чертежника в левом нижнем углу будущего
         | квадрата и поднято
    надо | нарисован квадрат с длиной стороны а, перо Чертежника
         | в исходной точке и поднято
нач
| опустить перо
| сместиться на вектор (а, 0)
| сместиться на вектор (0, а)
| сместиться на вектор (–а, 0)
| сместиться на вектор (0, –а)
| поднять перо
кон
```

Запись "**алг** квадрат (**арг вещ** а)" означает, что у алгоритма "квадрат" есть один аргумент (**арг**) *а*, который может быть произвольным числом.

Слово **вещ** — это сокращение от «вещественный», этим словом в информатике принято обозначать числа, которые могут быть целыми или дробными (в математике такие числа называют действительными). Это слово описывает **тип** аргумента. С понятием типа мы уже встречались при изучении баз данных. Напомним, что тип указывает, какие значения может принимать величина и какие действия можно с ней выполнять.

Позже мы познакомимся с алгоритмами, аргументы которых могут быть лишь целыми (**цел**) числами; с алгоритмами, аргументы которых вообще не являются числами; а также с алгоритмами, у которых, кроме аргументов, есть результаты (**рез**). Именно этим объясняется необходимость слов **арг** и **вещ** в заголовке алгоритма "квадрат".

14.2. Выполнение вспомогательного алгоритма с аргументами

Если в основном алгоритме написать вызов "квадрат (4)", то компьютер запомнит, что аргумент *а* равен 4, и при выполнении алгоритма "квадрат (**арг вещ** а)" скомандует *Чертежнику*:

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

Если же в основном алгоритме написать вызов "квадрат (6)", то компьютер запомнит, что аргумент *а* равен 6, и скомандует *Чертежнику*:

```
опустить перо
сместиться на вектор (6, 0)
сместиться на вектор (0, 6)
сместиться на вектор (–6, 0)
сместиться на вектор (0, –6)
поднять перо
```

14.3. Модель памяти компьютера

Поясним подробнее, как компьютер запоминает значения аргументов в своей памяти. Память компьютера удобно представлять в виде обычной классной доски, на которой можно записывать информацию, читать, стирать, записывать заново и т. д. Место, отводимое в памяти компьютера для хранения информации, удобно изображать прямоугольником, внутри которого

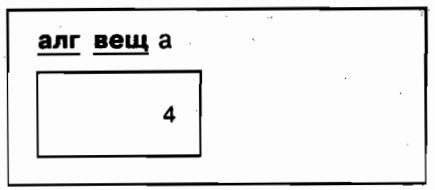


Рис. 60

записывается сама информация (значение аргумента). Такой прямоугольник мы будем называть ***ячейкой*** памяти.

При выполнении каждого алгоритма ему выделяется отдельный участок памяти, где создаются ячейки для аргументов. После окончания выполнения алгоритма эта память освобождается. Память алгоритма удобно изображать в виде большого прямоугольника, внутри которого находятся прямоугольники меньшего размера, соответствующие ячейкам.

Возможный вид памяти при исполнении алгоритма "квадрат" показан на рисунке 60 (здесь $a = 4$).

14.4. Алгоритмы с несколькими аргументами

Алгоритм может иметь несколько аргументов. Рассмотрим пример.

A 45

```
алг прямоугольник (арг вещ a, b)
    дано | перо Чертежника в левом нижнем углу А будущего
         | прямоугольника и поднято
    надо | нарисован прямоугольник со сторонами a и b, перо Чер-
         | тежника в точке А и поднято
нач
| опустить перо
| сместиться на вектор (a, 0)
| сместиться на вектор (0, b)
| сместиться на вектор (–a, 0)
| сместиться на вектор (0, –b)
| поднять перо
кон
```

При вызове такого алгоритма важно правильно задать порядок следования аргументов. Соответствие между аргументами в команде вызова и в заголовке алгоритма устанавливается по

порядку их следования. На рисунке 61, *а* показано изображение, полученное при вызове "прямоугольник (3, 7)", а на рисунке 61, *б* — изображение, полученное при вызове "прямоугольник (7, 3)".

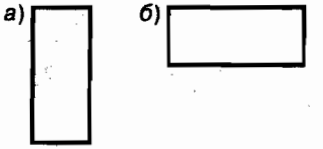


Рис. 61

14.5. Аргументы в заголовке цикла n раз

Аргументы алгоритма можно использовать в любых командах алгоритмического языка, где могут быть числа, в том числе в заголовке цикла n **раз**.

При помощи цикла n **раз** с аргументом можно составить алгоритмы для смещения *Робота* на заданное число клеток в нужном направлении, например:

A 46

```
алг влево на (арг цел n)
    дано | на поле Робота стен нет
    надо | Робот сместился на n клеток влево
нач
| нц n раз
|  влево
| кц
кон
```

Аналогично можно составить алгоритмы "вниз на (**арг цел** n)", "вверх на (**арг цел** n)" и "вправо на (**арг цел** n)". В дальнейшем мы часто будем пользоваться этими алгоритмами как вспомогательными.

Обратите внимание, что без помощи цикла **n раз** алгоритм А46 составить нельзя. Таким образом, применение аргументов открывает новые возможности в использовании этой простой конструкции: цикл n **раз** с аргументом не только сокращает запись, но и позволяет решать задачи, для которых невозможен простой линейный алгоритм.

14.6. Закрашивание прямоугольника

В § 9 мы решали задачу о закрашивании *Роботом* прямоугольника. Тогда мы вынуждены были указывать в алгоритме конкретные размеры прямоугольника. Используя аргументы, мы мо-

жем переписать этот алгоритм так, чтобы при его выполнении закрашивался прямоугольник произвольного размера.

А 47

```
алг закрасить прямоугольник (арг цел m, n)
    дано | на поле Робота стен нет
    надо | закрашен прямоугольник размером m*n, Робот в исход-
         | ном положении
нач
| нц n раз
| |  закрасить ряд (m)
| |  вниз
| кц
| вверх на (n)
кон
```

А 48

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

14.7. Заголовок алгоритма с аргументами

Как мы уже знаем, заголовок алгоритма описывает условие задачи, а тело алгоритма — ее решение. Чтобы записать заголовок алгоритма, обычно достаточно внимательно изучить условие, не думая пока о решении.

При построении алгоритмов с аргументами важно точно определить количество аргументов и их типы. Для этого нужно изучить условие задачи и выделить в нем ту информацию, которую необходимо задать, прежде чем приступать к решению. Этой информации будут соответствовать аргументы алгоритма.

Например, в задаче «построить квадрат» такой дополнительной информацией была сторона квадрата, поэтому у алгоритма появился один аргумент.

В общем случае переменным в условии задачи соответствуют аргументы в заголовке алгоритма.

ЗАДАЧИ И УПРАЖНЕНИЯ

1. Какую последовательность команд выполнит *Чертежник* и что будет нарисовано при вызовах: а) квадрат (0); б) квадрат (–1)?

2. Составьте алгоритм "прямоугольник (**арг вещ** x, y, a, b)", который рисует прямоугольник с длинами сторон *a* и *b*, начиная и заканчивая в углу — точке (*x*, *y*) (рис. 62).

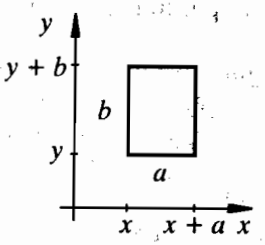
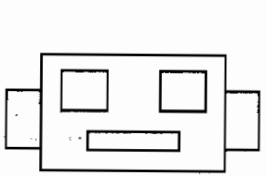


Рис. 62

3. Используя алгоритм "прямоугольник" (упр. 2), составьте алгоритмы рисования робота и собачки (рис. 63).


а) Робот

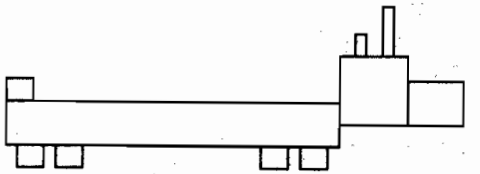
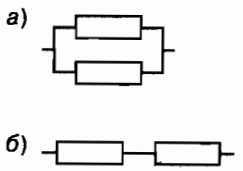

б) Собачка

Рис. 63

4. Составьте алгоритмы рисования схем (рис. 64).



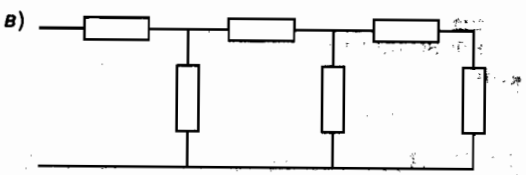


Рис. 64

5. Придумайте какую-нибудь картинку, составленную из прямоугольников. Напишите алгоритм для рисования этой картинки.

6. Сколько клеток будет закрашено и сколько команд компьютер выдаст *Роботу* при выполнении вызова:
а) закрасить прямоугольник (1, 1);
б) закрасить прямоугольник (0, 11);
в) закрасить прямоугольник (9, 0);
г) закрасить прямоугольник (9, 11)?

7. Опишите, как будет выполняться вызов "закрасить прямоугольник (3, 3)" в ситуациях, изображенных на рисунке 65.

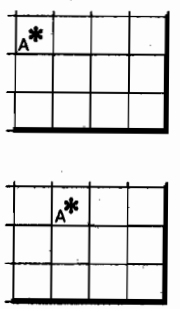


Рис. 65

8. Измените алгоритмы "закрасить ряд" (А48) и "закрасить прямоугольник" (А47) так, чтобы при вызове "закрасить прямоугольник (3, 3)" в ситуации на рисунке 65, *б* отказа не возникало, а оказался закрашенным квадрат 3×3 клетки.

9. Нарисуйте результат выполнения алгоритма:

А 49

а) **алг** тоннель
нач
| квадрат (10); сместиться на вектор (1, 1)
| квадрат (7); сместиться на вектор (1, 1)
| квадрат (4); сместиться на вектор (1, 1)
| квадрат (1)
кон

А 50

б) **алг** спираль
нач
| опустить перо
| виток (1); виток (3); виток (5); виток (7); виток (9)
| поднять перо
кон

А 51

алг виток (**арг вещ** a)
нач
| сместиться на вектор (a, 0)
| сместиться на вектор (0, –a)
| сместиться на вектор (–a – 1, 0)
| сместиться на вектор (0, a + 1)
кон

10. Что нарисует *Чертежник* при выполнении алгоритма "спираль" (А50), если в алгоритме "виток" (А51) всюду заменить 1 на 2?

11. Измените алгоритм "виток" (А51) так, чтобы спираль в алгоритме А50 раскручивалась против часовой стрелки.

12. Составьте алгоритм рисования спирали, изображенной на рисунке 66.

13. Измените ваше решение упражнения 12 так, чтобы расстояние между витками при каждом новом витке увеличивалось.

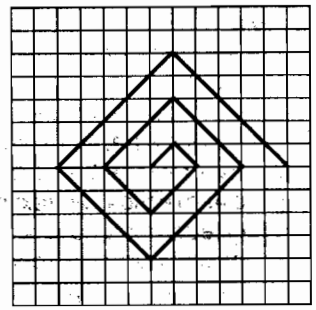


Рис. 66

14. Нарисуйте результат выполнения алгоритма "орнамент":

А 52

алг орнамент
дано | перо Чертежника в левом верхнем углу будущего
| орнамента размером 12×12 и поднято
надо | нарисован орнамент, перо в левом нижнем углу
| и поднято
нач
| ряд; ряд; ряд
кон

А 53

алг ряд
дано | перо Чертежника в левом верхнем углу будущего ряда
| размером 12×4 и поднято
надо | нарисован ряд, перо в левом нижнем углу ряда
| и поднято
нач
| фрагмент; фрагмент; фрагмент
| сместиться на вектор (–12, –4)
кон

А 54

алг фрагмент
дано | перо Чертежника в левом верхнем углу будущего
| фрагмента размером 4×4 и поднято
надо | нарисован фрагмент, перо в правом верхнем углу
| и поднято
нач
| опустить перо
| сместиться на вектор (2, –2)
| сместиться на вектор (–2, –2)
| поднять перо; сместиться на вектор (4, 0); опустить перо
| сместиться на вектор (0, 1)
| сместиться на вектор (–2, 0)
| сместиться на вектор (0, 2)
| сместиться на вектор (2, 0)
| сместиться на вектор (0, 1)
| поднять перо
кон

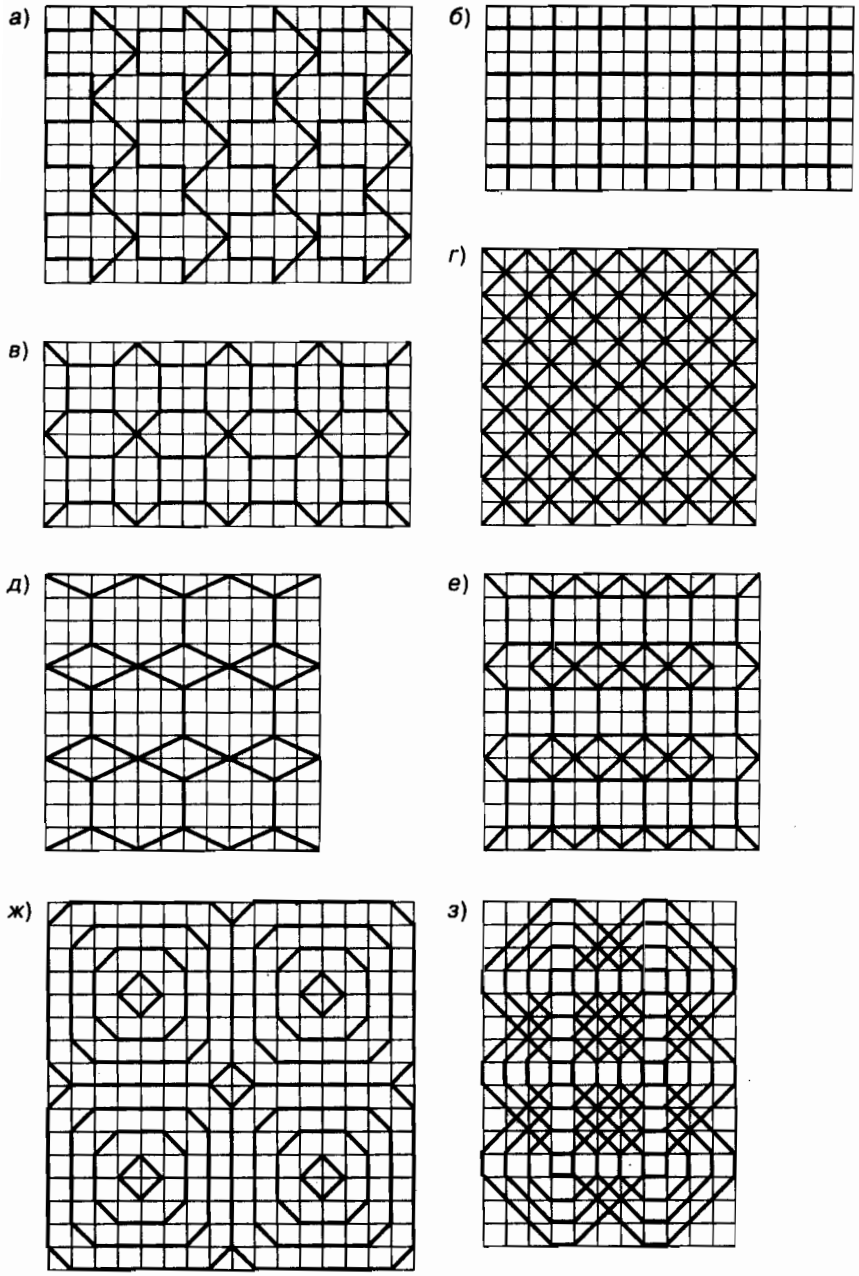


Рис. 67

15. По образцу упражнения 14 составьте алгоритмы рисования орнаментов (рис. 67).

16. Составьте алгоритм "горизонтальная ломаная (**арг цел** n, **арг вещ** a)", рисующий с помощью *Чертежника* ломаную линию с $2n$ звеньями, показанную на рисунке 68.

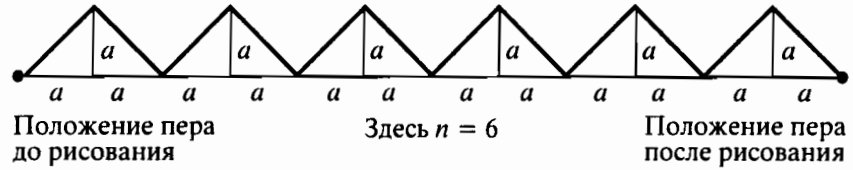


Рис. 68

17. Составьте алгоритм "вертикальная ломаная (**арг цел** n, **арг вещ** a)", рисующий ломаную из упражнения 16, повернутую вокруг начального положения пера на 90° по часовой стрелке.

18. Используя алгоритмы из упражнений 16 и 17 как вспомогательные, составьте алгоритмы, рисующие:

а) m горизонтальных ломаных с $2n$ звеньями одна под другой на расстоянии b друг от друга;

б) m вертикальных ломаных с $2n$ звеньями одна под другой на расстоянии b друг от друга.

19. Компьютер выполнил последовательность команд:

горизонтальная ломаная (5, 1)
вертикальная ломаная (7, 1)
горизонтальная ломаная (5, –1)
вертикальная ломаная (7, –1)

Что нарисовал *Чертежник*?

20. Составьте алгоритм с целыми аргументами m и n, который с помощью *Робота* закрашивает клетки, отмеченные на рисунке 69.

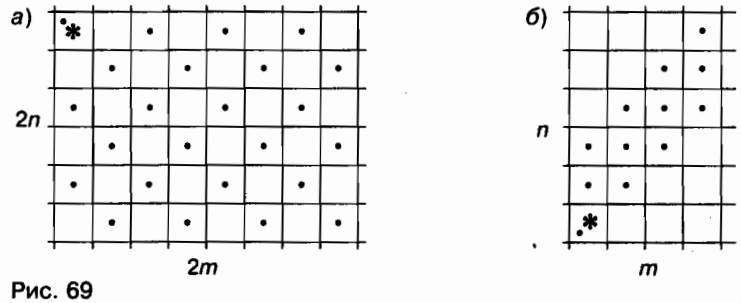


Рис. 69

15 Арифметические выражения и правила их записи

15.1. Арифметические выражения в алгоритмическом языке

В алгоритмическом языке можно использовать не только числа, но и числовые и алгебраические выражения (формулы). В информатике эти выражения называются ***арифметическими***. Правила алгоритмического языка позволяют при записи алгоритма всюду, где можно написать число, ввести и произвольное арифметическое выражение.

Рассмотрим пример:

А 55

```
алг нарисовать М размером (арг вещ a, b)
  дано | перо Чертежника в точке А (рис. 70) и поднято
  надо | нарисована буква М ширины a и высоты b,
       | перо поднято и находится в точке Б (рис. 70)
нач
  опустить перо
  сместиться на вектор (0, b)
  сместиться на вектор (a/2, -b/2)
  сместиться на вектор (a/2, b/2)
  сместиться на вектор (0, -b)
  поднять перо
  сместиться на вектор (a/2, 0)
кон
```

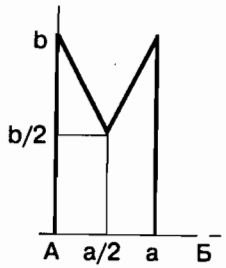


Рис. 70. Буква с переменными размерами

Знак «/» здесь означает деление.

Таким образом, запись «a/2» означает $\frac{a}{2}$,

«(a + b)/c» означает $\frac{a+b}{c}$ и т. д.

15.2. Выражения вычисляет компьютер

При выполнении вызова "нарисовать М размером (3, 5)" компьютер запомнит, что $a = 3, b = 5$, вычислит все выражения и скомандует *Чертежнику*:


```
опустить перо
сместиться на вектор (0, 5)
сместиться на вектор (1.5, -2.5)
сместиться на вектор (1.5, 2.5)
сместиться на вектор (0, -5)
поднять перо
сместиться на вектор (1.5, 0)
```

Чертежник никаких вычислений не производит — он лишь выполняет последовательно поступающие команды с конкретными числовыми аргументами.

15.3. Правила записи арифметических выражений в алгоритмическом языке

Арифметические выражения[1] в алгоритмическом языке должны быть записаны в так называемой ***линейной*** записи согласно следующим правилам:

— выражение должно быть записано в виде линейной цепочки символов (вместо x_1 и y_0 надо писать $x1, y0$);

— для обозначения операции умножения используется звездочка (*), для операции деления — косая черта (/), для операции возведения в степень — две звездочки (**);

— нельзя опускать знаки операций, например писать $4a$. Для записи произведения чисел 4 и a надо писать $4*a$;

— аргументы функций (sin, cos и др.), как и аргументы вспомогательных алгоритмов, записываются в круглых скобках, например sin (x), cos ($4*a$);

— для изменения порядка действий используются круглые скобки.

Линейная запись позволяет вводить выражения в память компьютера, последовательно нажимая на соответствующие клавиши на клавиатуре.

[1] В данном разделе описаны все возможности алгоритмического языка по записи выражений, в том числе тригонометрических и логарифмов. Все эти выражения входят в материал старших классов, а пока можно пропустить то, что непонятно.

15.4. Операции и стандартные функции алгоритмического языка

Основные операции и функции алгоритмического языка приведены ниже.

ТАБЛИЦА 12. **Операции и стандартные функции алгоритмического языка**

Название операции или функции	Форма записи
сложение	x + y
вычитание	x – y
умножение	x*y
деление	x/y
возведение в степень	x**y
корень квадратный $\sqrt{x}$	sqrt(x)
абсолютная величина $\|x\|$	abs(x)
знак числа (–1, 0, или 1)	sign(x)
синус sin x	sin(x)
косинус cos x	cos(x)
тангенс tg x	tg(x)
котангенс ctg x	ctg(x)
арксинус arcsin x	arcsin(x)
арккосинус arccos x	arccos(x)
арктангенс arctg x	arctg(x)
арккотангенс arcctg x	arcctg(x)
натуральный логарифм ln x	ln(x)
десятичный логарифм lg x	lg(x)
степень числа e ($e \approx 2{,}718281$) e^x	exp(x)
минимум из чисел x и y	min(x, y)
максимум из чисел x и y	max(x, y)
остаток от деления x на y (x, y — целые)	mod(x, y)
частное от деления x на y (x, y — целые)	div(x, y)
целая часть x, т. е. максимальное целое число, не превосходящее x	int(x)
случайное число от 0 до x (см. § 25)	rnd(x)

15.5. Порядок действий в арифметических выражениях

При вычислении арифметических выражений компьютер выполняет действия в следующем порядке:

1) вычисляются выражения в скобках (в том числе аргументы функций); порядок действий внутри скобок определяется теми же правилами (т. е. сначала вычисляются скобки внутри скобок и т. д.);

2) вычисляются значения функций;

3) *справа налево* выполняются возведения в степень;

4) слева направо выполняются умножение и деление;

5) слева направо выполняются сложение и вычитание.

Возведение в степень справа налево означает, что запись $a^{**}b^{**}c$ следует понимать как a^{b^c}, но не как $(a^b)^c$.

Умножение, деление, сложение и вычитание выполняются слева направо. Например, запись $a + b - c$ означает $(a + b) - c$, но не $a + (b - c)$.

Примеры записи арифметических выражений на алгоритмическом языке

Выражение	Линейная запись
$\frac{-1}{x^2}$	–1/x**2
$\frac{a}{bc}$	a/(b*c)
$\frac{a}{b}c$	a/b*c или (a/b)*c
$2^{2^{2^n}}$	2**2**2**n или 2**(2**(2**n))
x^{y^z}	x**y**z или x**(y**z)
$\frac{-b + \sqrt{b^2 - 4ac}}{2a}$	(–b + sqrt (b**2 – 4*a*c))/(2*a)
$\sqrt{p(p-a)(p-b)(p-c)}$	sqrt (p*(p – a)*(p – b)*(p – c))
$\frac{a+b+c}{2}$	(a + b + c)/2
$\sqrt{a^2 + b^2 + 2ab\cos\gamma}$	sqrt(a**2 + b**2 – 2*a*b*cos(γ))
$\frac{ad + bc}{bd}$	(a*d + b*c)/(b*d)
$\frac{\operatorname{tg}\alpha}{\sqrt{1 + \operatorname{tg}^2\alpha}}$	tg(α)/sqrt (1 + tg (α)**2)
$\sin\alpha\cos\beta + \cos\alpha\sin\beta$	sin (α)*cos (β) + cos (α)*sin(β)

ЗАДАЧИ И УПРАЖНЕНИЯ

1. Вычислите значение выражения, записанного на алгоритмическом языке:

а) 24/(3*4) – 24/3/4 + 24/3*4;
б) 40/(4*5) – 40/4/5 + 40/4*5;
в) (2 + 3*4)/2 + 5 – (2 + sqrt (4));
г) 60/(23 – (2 + 3*sqrt (5 – abs (1 – 3))));
д) 71 + abs ((16 – 7*2)/2) – sqrt (sqrt (225)).

2. Переведите из линейной записи в обычную:

а) a/b/c;
б) a*b/c;
в) a/b*c;
г) a/b**c;
д) a + b/c;
е) (a + b)/c;
ж) a/b**c**d;
з) 1/(1 + x*x);
и) 1/(1 + x**2).

3. Переведите из линейной записи в обычную:

а) 1/sqrt (1 + x**2);
б) sqrt (x**2 + y**2);
в) x**(1/3);
г) x**(–1/3);
д) 1/x**(1/3);
е) sin (x)**2 + sin (y)**2;
ж) sin (x**2) + sin (y**2);
з) a + b/c + d;
и) (a + b)/(c + d);
к) a/sin (A).

4. Переведите из линейной записи в обычную:

а) sqrt (tg (A + B))/sqrt (tg (A – B));
б) 1/2*a*b*sin (C);
в) sqrt (b**2 + c**2 + 2*b*c*cos (A))/2;
г) 2*b*c*cos (A/2)/(b + c);
д) sqrt ((p – a)*(p – b)*(p – c)*p);
е) 4*R*sin (A/2)*sin (B/2)*sin (C/2);
ж) (a*x + b)/(c*x + d);
з) sqrt (a*x**2 + b*x + c);
и) arctg (x/sqrt (1 – x**2));
к) 2*sin ((alfa + beta)/2)*cos ((alfa – beta)/2).

5. Запишите по правилам алгоритмического языка следующие выражения:

а) $\sqrt{x_1^2 + x_2^2}$;
б) $x_1x_2 + x_1x_3 + x_2x_3$;
в) $v_0t + \frac{at^2}{2}$;
г) $\frac{mv^2}{2} + mgh$;
д) $\frac{1}{R_1} + \frac{1}{R_2}$;
е) $mg \cos \alpha$;
ж) $2\pi R$;
з) $b^2 - 4ac$;
и) $\gamma\frac{m_1m_2}{r^2}$;
к) $|x| + |x + 1|$;
л) I^2R;
м) $ab \sin C$;
н) $\sqrt{a^2 + b^2 - 2ab\cos C}$;
о) $\sin x \cos y + \sin y \cos x$;
п) $\frac{ad + bc}{bd}$;
р) $a \sin \alpha + b \cos \beta$;
с) $\sqrt{1 - \sin 2x}$;
т) $-\frac{1}{\sqrt{ax^2 + bx + c}}$;
у) $-\frac{\sqrt{x + 1} - \sqrt{x - 1}}{2\sqrt{x}}$;
ф) $|1 - |x||$.

16 Величины в алгоритмическом языке. Команда присваивания

16.1. Измерение радиации и температуры

В § 10 мы рассмотрели 10 из 12 команд-вопросов *Робота*. Познакомимся с оставшимися двумя командами.

По команде температура *Робот* измеряет температуру в клетке, где он находится, и сообщает результат. Температура измеряется в градусах Цельсия, она может меняться в пределах от –100 °С до 100 °С.

По команде радиация *Робот* измеряет и сообщает уровень радиации в текущей клетке. Радиация измеряется в условных единицах и изменяется от 0 до 10.

На пульте управления *Роботом* (см. рис. 15 на с. 64) этим командам соответствует нижняя секция. При нажатии кнопки радиация или температура на экране, расположенном в этой части пульта, появляется число, означающее соответственно радиацию или температуру в клетке, где находится *Робот*.

Команды температура и радиация можно использовать в алгоритме в любом месте, где допускается вещественное число.

Рассмотрим пример. *Робот* стоит перед входом в тупик неизвестной длины. Необходимо пометить (закрасить) в тупике все клетки, радиация в которых превышает заданный

опасный уровень *a*, и вернуть *Робота* в исходное положение (рис. 71).

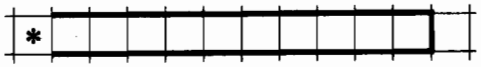


Рис. 71

В условии задачи есть параметр — следовательно, в алгоритме будет аргумент.

A 56

```
алг разведка в тупике (арг вещ а)
    дано | Робот стоит левее горизонтального тупика
    надо | закрашены все клетки тупика, в которых уровень
         | радиации выше а, Робот в исходном положении
нач
|   нц пока справа свободно
|   | вправо
|   | если радиация > а
|   |   то закрасить
|   | все
|   кц
|   нц пока сверху стена
|   | влево
|   кц
кон
```

16.2. Компьютер запоминает информацию

Изменим условие задачи. Пусть опасный уровень радиации заранее не задан, но известно, что он равен радиации в клетке перед тупиком. Иными словами, необходимо закрасить все клетки тупика, уровень радиации в которых выше, чем в клетке, где в начальный момент находится *Робот*.

При непосредственном управлении *Роботом* можно поступить так. Узнаем у *Робота* радиацию в начальной клетке и запомним ее. Дальше можно действовать по алгоритму А56, сравнивая радиацию в каждой клетке тупика с запомненным значением.

Чтобы добиться того же результата с помощью программного управления, необходимо, измерив радиацию в начальной точке, каким-то образом запомнить ее, т. е. выделить для этой информации ячейку памяти и занести в нее нужное значение.

На алгоритмическом языке это можно записать так:

A 57

```
алг разведка в тупике 2
    дано | Робот стоит левее горизонтального тупика
    надо | закрашены все клетки тупика, в которых уровень
         | радиации выше, чем в исходном положении Робота
         | Робот в исходном положении
нач вещ у  | создать ячейку с именем у
|   у := радиация | запомнить значение радиации
|   нц пока справа свободно
|   | вправо
|   | если радиация > у
|   |   то закрасить
|   | все
|   кц
|   нц пока сверху стена
|   | влево
|   кц
кон
```

Для организации работы с памятью компьютера в алгоритмическом языке используются величины. В алгоритме А57 использована одна величина *у*. Выделение ячейки памяти для хранения информации происходит при *описании* величины (**вещ** у), а занесение в ячейку нужной информации происходит при выполнении *команды присваивания* (у:=радиация). Подробнее о работе с величинами рассказано в разделах 16.4 — 16.8.

16.3. Компьютер выполняет подсчет

Еще усложним условие задачи. Пусть опасные клетки надо отметить не в тупике, а на участке свободного пространства между *Роботом* и стеной (рис. 72). Изменение ситуации не мешает выполнить разметку уже известным способом, но при возвращении в исходное положение возникает проблема: как узнать, в какой клетке остановиться?

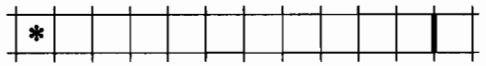


Рис. 72

При непосредственном управлении *Роботом* мы можем считать шаги, которые он делает при движении к стене. Когда *Робот* подойдет к стене, мы будем знать пройденное им расстоя-

ние и сможем приказать ему сделать такое же число шагов в обратную сторону.

При программном управлении необходимо, чтобы считать шаги мог компьютер. Как это делается, показано в алгоритме:

A 58

```
алг разведка в чистом поле
    дано | где-то справа от Робота есть стена
    надо | закрашены все клетки между Роботом и стеной,
         | в которых уровень радиации выше, чем в исходном
         | положении Робота
         | Робот в исходном положении
нач вещ y | создать ячейку с именем y
|   цел n | создать ячейку n для подсчета количества шагов
|   y := радиация | запомнить значение радиации
|   n := 0 | обнулить счетчик
|   нц пока справа свободно
|       вправо
|       n := n + 1 | увеличить на 1 счетчик количества шагов
|       если радиация > y
|       |  то закрасить
|       все
|   кц
|   влево на (n)
кон
```

В этом алгоритме использованы две величины. Величина *y* используется для хранения «опасного» уровня радиации. Этот уровень запоминается в исходном положении *Робота* и в дальнейшем многократно используется. Величина *n* играет роль счетчика. В начальный момент счетчик «обнуляется» командой n:=0, а затем многократно увеличивается командой n:=n+1.

16.4. Величины и их характеристики

При решении многих задач надо уметь запоминать, изменять и использовать информацию в памяти компьютера (например, в рассмотренной задаче это информация об опасном уровне радиации и числе сделанных вправо шагов). Для этого в алгоритмическом языке используются ***величины***.

Термин «величина» заимствован из математики и физики, поскольку в алгоритмах для решения математических или физических задач величины алгоритмического языка соответствуют математическим или физическим величинам.

Каждая величина имеет ***имя, тип, вид*** и ***значение***.

Имя величины служит для обозначения величины в алгоритме. Именем величины в алгоритмическом языке может быть любое слово (кроме служебных слов самого языка) или даже несколько слов. В имя могут входить русские и латинские буквы и даже цифры (правда, начинаться с цифры имена не могут), а также пробелы и знаки подчеркивания. Вот несколько примеров допустимых имен: *s, R, слово 1, длинное имя*.

Тип величины показывает, какие значения может принимать величина и какие операции можно с ней выполнять. Пока мы будем пользоваться двумя типами: вещественным (**вещ**) и целым (**цел**). Значением величины вещественного типа может быть любое число, а целого — только целое. Необходимость различать эти типы связана с тем, что они по-разному обрабатываются компьютером: все вычисления с целыми числами выполняются точно, а с вещественными — приближенно (хотя и с высокой точностью). Позже мы познакомимся с величинами некоторых других типов.

Вид величины показывает ее информационную роль в алгоритме. ***Аргументы*** содержат исходную информацию, необходимую для работы алгоритма, а ***промежуточные величины*** предназначены для хранения текущей информации, которую обрабатывает алгоритм. Позднее мы познакомимся с другими видами величин.

Во время выполнения алгоритма в каждый конкретный момент величина имеет какое-то ***значение*** (например, 22 или –107) либо ***не определена***.

Имя, тип и вид величины можно однозначно определить по тексту алгоритма. Это ***статические*** характеристики величины. Значение определяется только во время выполнения. Это ***динамическая*** характеристика.

16.5. Описание величин

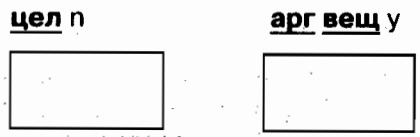
Чтобы компьютер мог работать с величиной, нужно указать ее вид, тип и имя. Такое указание называется ***описанием*** величины.

Аргументы описывают в заголовке алгоритма, явно указывая вид, тип и имя, например **арг вещ** y.

Промежуточные величины описывают в теле алгоритма после слова **нач**. Их вид специально никак не обозначается, так как именно описание в теле алгоритма говорит о том, что это промежуточная величина.

16.6. Модель памяти компьютера

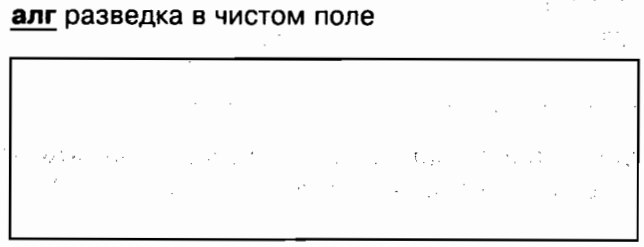
Для запоминания информации в компьютере имеется ***память***. Напомним (см. 14.3), что память компьютера удобно представлять в виде классной доски, на которой можно записывать информацию, читать, стирать, записывать заново и т. д. Место, отводимое в памяти компьютера для каждой величины, удобно изображать в виде прямоугольника. Значение величины (если она определена) записывается внутри прямоугольника, а тип и имя указываются сверху:



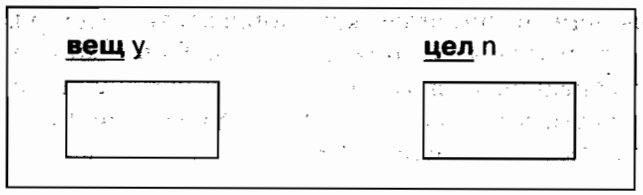
Если величина не определена, то в прямоугольнике ничего не пишется.

Рассмотрим работу памяти на примере алгоритма А58.

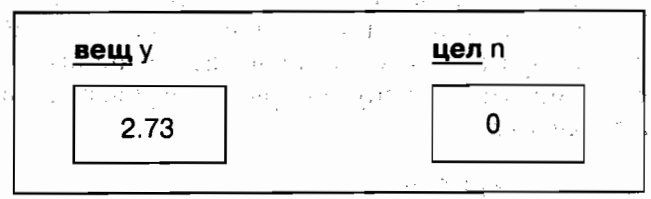
В начале выполнения алгоритма компьютер выделит для него часть памяти:



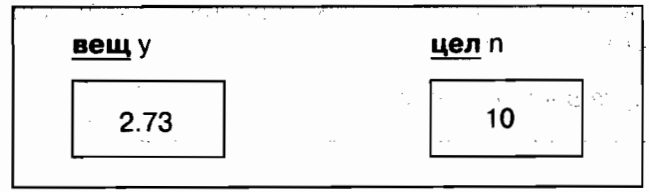
Встретив после слова **нач** описания **вещ** y и **цел** n, компьютер отведет внутри памяти алгоритма место для хранения двух величин: первая имеет вещественный тип и имя *y*, вторая — целый тип и имя *n*.



После первых двух команд алгоритма эти величины получат значения. В ячейке *n* будет 0, а в ячейке *y* — значение радиации в клетке с *Роботом*.



При дальнейшем исполнении алгоритма значение величины *y* уже не изменится, а *n* будет увеличиваться по мере продвижения *Робота* в тупик. В конце выполнения алгоритма память может выглядеть так:



Когда выполнение алгоритма заканчивается, память освобождается, значения промежуточных величин пропадают.

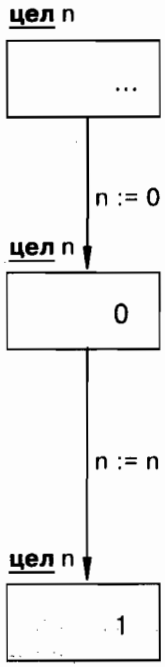
16.7. Команда присваивания

Для запоминания или изменения значения величины в алгоритмическом языке есть специальная команда — ***команда присваивания***, которая записывается в виде:

имя величины := выражение

Знак «:=» (двоеточие, а потом равенство) называется знаком ***присваивания*** и читается как «присвоить» (например, команда n := e читается: «*n* присвоить *e*»). При выполнении команды присваивания компьютер сначала вычисляет записанное в правой части выражение (заменяя имена величин на их значения), а потом полученное значение выражения записывает в память.

16.8. Примеры использования команды присваивания



При выполнении команды присваивания n := 0 компьютер запишет значение выражения (т. е. 0) внутрь прямоугольника «n» (старое значение, каково бы оно ни было, будет затерто).

Таким образом, после выполнения команды n := 0 величина *n* будет иметь значение 0.

Если теперь выполнить команду n := n + 1, то компьютер сначала вычислит значение выражения *n* + 1, т. е. заменит имя *n* на значение 0 и вычислит 0 + 1 = 1. После этого компьютер сотрет старое значение величины *n* и запишет новое (1).

Другими словами, после выполнения команды n := n + 1 значение величины *n* будет увеличено на 1.

16.9. Еще один пример алгоритма, работающего с величинами

Рассмотрим следующую задачу. *Робот* расположен в клетке над горизонтальной стеной неизвестной длины. Надо переместить *Робота* на клетку вниз — «сквозь стену» (рис. 73).

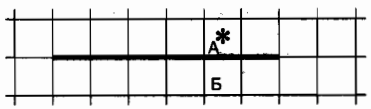


Рис. 73

Поскольку *Робот* сквозь стену проходить не умеет, стену нужно обойти. Сделать это можно так. Пока ниже *Робота* стена, будем двигать его вправо и по дороге считать число шагов (число команд вправо). Как только стена кончится, сместим *Робота* вниз и начнем двигать его обратно (влево). Тут нам и понадобится запомненная ранее информация о числе шагов вправо: скомандуем *Роботу* влево столько же раз, сколько было сделано шагов вправо, и он окажется в точке Б.

Запишем теперь алгоритм "вниз сквозь стену", используя для подсчета сделанных вправо шагов целочисленную величину *n*:

A 59

```
алг вниз сквозь стену
    дано | Робот над горизонтальной стеной, других стен нет
    надо | Робот под стеной, на клетку ниже исходного положения
нач цел n
| n := 0
| нц пока снизу стена
| | вправо; n := n + 1
| кц
| утв | Робот вышел за край стены,
| | n = число сделанных вправо шагов
| вниз
| влево на (n)
кон
```

При выполнении этого алгоритма компьютер сначала присвоит величине *n* значение 0 (команда n := 0). Затем в цикле компьютер будет командовать *Роботу* вправо и тут же увеличивать значение *n* на 1 (команда n := n + 1). После каждого выполнения тела цикла значение *n* окажется равным числу сделанных *Роботом* шагов вправо. После окончания цикла значением *n* будет общее число сделанных вправо шагов. Поэтому, когда при выполнении вспомогательного алгоритма "влево на (n)" компьютер *n* раз скомандует влево, *Робот* окажется в точности на клетку ниже исходного положения (рис. 74).

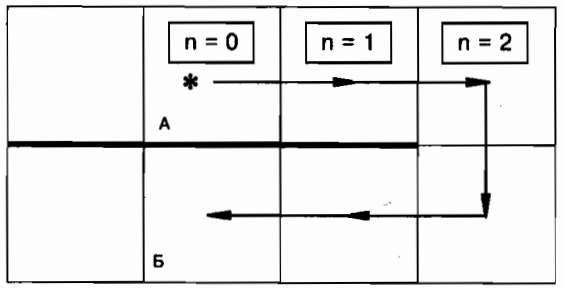


Рис. 74

16.10. Рисование параболы

Рассмотрим следующую задачу: изобразить с помощью *Чертежника* график функции (параболу) $y = x^2$. Условие этой задачи нуждается в уточнении. Во-первых, нарисовать весь график нельзя — ведь он бесконечен. Поэтому будем рисовать только его часть, например от $x = 0$ до $x = 2$. Во-вторых, *Чертежник* не умеет рисовать ничего, кроме отрезков, а график функции $y = x^2$ — это кривая. Поэтому параболу придется заменить ломаной. Если вершины ломаной лежат на параболе и звенья ломаной очень короткие, то ломаная на вид почти не отличается от параболы. На рисунке 75, *а* ломаная имеет 4 звена, на рисунке 75, *б* — 10 звеньев, на рисунке 75, *в* — 100 звеньев.

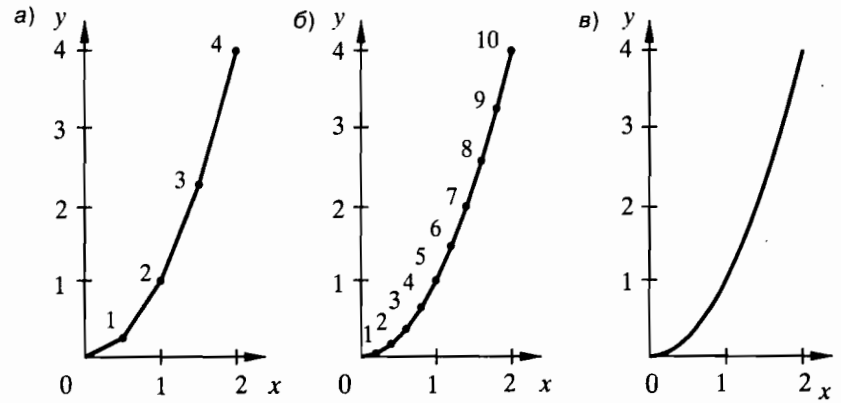


Рис. 75

В общем случае с помощью *Чертежника* можно изобразить параболу на некотором участке (от a до b) в виде ломаной из какого-то числа звеньев (n).

Будем рисовать ломаную слева направо, начиная от точки $(a, a^{**}2)$ и кончая точкой $(b, b^{**}2)$. От вершины к вершине будем перемещаться с помощью команды *Чертежника* сместиться в точку.

Используем вещественную величину x для запоминания x-координаты очередной вершины ломаной.

Разобьем отрезок от a до b на n равных частей длины $d = (b - a)/n$ (рис. 76).

При рисовании очередного звена ломаной величину x надо просто увеличивать на d — это можно сделать с помощью команды присваивания x := x + d:

алг парабола (**арг вещ** a, b, **арг цел** n) | A 60
  дано n > 0 | перо Чертежника поднято
  надо | нарисован график функции y = x**2 на участке от a до b
  | в виде ломаной из n звеньев; перо в точке (b, b**2)
  | и поднято
нач вещ x, d
| x := a; d := (b – a)/n
| сместиться в точку (a, a**2)
| опустить перо
| **нц** n **раз**
| | x := x + d
| | сместиться в точку (x, x**2)
| **кц**
| поднять перо
кон

При выполнении вызова "парабола (0, 2, 10)" компьютер выдаст *Чертежнику* последовательно 13 команд:

сместиться в точку (0, 0)
опустить перо
сместиться в точку (0.2, 0.04)
сместиться в точку (0.4, 0.16)
сместиться в точку (0.6, 0.36)
сместиться в точку (0.8, 0.64)
сместиться в точку (1.0, 1.00)
сместиться в точку (1.2, 1.44)
сместиться в точку (1.4, 1.96)
сместиться в точку (1.6, 2.56)
сместиться в точку (1.8, 3.24)
сместиться в точку (2.0, 4.00)
поднять перо

и *Чертежник* нарисует параболу на участке от 0 до 2 в виде ломаной из 10 звеньев.

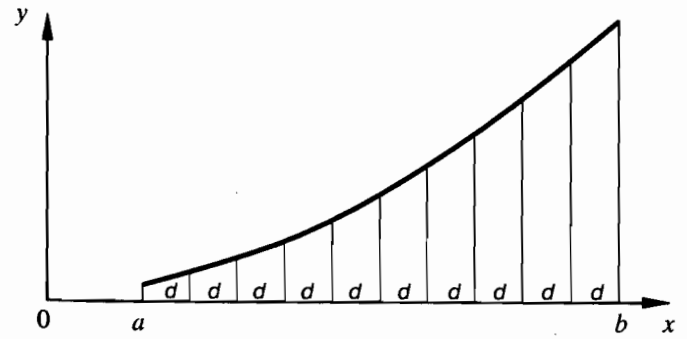


Рис. 76

ЗАДАЧИ И УПРАЖНЕНИЯ

1. Значение величины *x* равно 3. Чему оно будет равно после выполнения команды:

а) x := 5; б) x := x + 5; в) y := x?

2. После выполнения команды

а) x := x +5; б) x := –x; в) y := x

или серии команд

г) y := 1; x := x + y; д) y := x; x := y

значение величины *x* стало равно 3. Чему было равно значение величины *x* до выполнения в каждом из этих случаев?

3. После выполнения команды присваивания x := x + y значение величины *x* равно 3, а значение *y* равно 5. Чему были равны значения величин *x* и *y* до выполнения команды?

4. Значение величины *x* равно *a*, значение *y* равно *b*. После выполнения каких из указанных ниже последовательностей команд значения величин *x* и *y* поменяются, т. е. *x* станет равно *b*, а *y* станет равно *a*?

а) x := y
y := x

б) t := x
x := y
y := t

в) x := x + y
y := x – y
x := x – y

г) t := x
y := t
x := y

5. Опишите, что произойдет при выполнении алгоритма "вниз сквозь стену" для начальных положений *Робота*, указанных на рисунке 77.

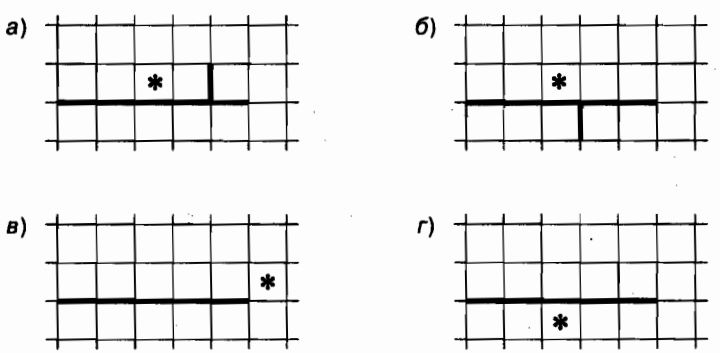


Рис. 77

6. Дан алгоритм:

А 61

```
алг график (арг вещ b, арг цел n)
    дано n > 0 | перо Чертежника поднято
    надо | нарисован график некоторой функции на участке от 0
         | до b в виде ломаной из n звеньев; перо в конце
         | графика и поднято
нач вещ x, d, v
| x := 0; d := b/n
| сместиться в точку (0, 0)
| опустить перо
| нц n раз
| | v := 2*x*d + d*d; x := x + d
| | сместиться на вектор (d, v)
| кц
| поднять перо
кон
```

а) Как будет выглядеть память ЭВМ при выполнении вызова "график (2, 10)"?

б) Что нарисует *Чертежник* при выполнении этого вызова?

7. *Робот* находится перед входом в тупик, идущий вправо. Составьте алгоритм, при выполнении которого с помощью *Чертежника* будет нарисован график радиоактивности в тупике (считая, что единица масштаба по оси *Ox* равна размеру клетки на поле *Робота*).

8. Составьте алгоритмы со следующими заголовками:

а)
```
алг вниз до стены закрасить и вернуться
    дано | где-то ниже Робота есть стена
    надо | Робот дошел до этой стены, закрасил клетку
         | и вернулся в исходное положение
```

б)
```
алг закрасить клетку в правом нижнем углу
    дано | Робот где-то внутри прямоугольника,
         | огороженного стенами, других стен нет
    надо | Робот закрасил клетку в правом нижнем углу
         | прямоугольника и вернулся в исходное положение
```

в) **алг** отойти вдвое дальше от левой стены

дано | где-то левее Робота есть стена, других стен нет
надо | Робот отошел вправо на расстояние от стены
| вдвое большее, чем исходное

г) **алг** симметрия

дано | где-то ниже Робота есть горизонтальная стена
| длиной в одну клетку, других стен нет
надо | Робот оказался в положении, симметричном
| исходному относительно стены

д) **алг** симметрия

дано | где-то ниже Робота есть горизонтальная стена, ко-
| торая и вправо и влево кончается, других стен нет
надо | Робот оказался в положении, симметричном
| исходному относительно стены

е) **алг** обойти прямоугольное препятствие

дано | Робот над прямоугольником, огороженным стена-
| ми, других стен нет
надо | Робот под прямоугольником на той же вертикали

ж) **алг** вниз сквозь бесконечную стену

дано | Робот над горизонтальной стеной, в одну сторону
| уходящей в бесконечность, других стен нет
надо | Робот под стеной на клетку ниже исходного
| положения

9. *Робот* находится в левом верхнем углу прямоугольника, огороженного стенами. Составьте алгоритм, после выполнения которого *Робот* будет стоять в клетке с минимальным уровнем радиации.

17 Алгоритмы с результатами

17.1. Простейший пример алгоритма с результатами

Как мы уже знаем, для задания исходной информации в алгоритме используются аргументы. Но алгоритм может не только получать информацию, но и отдавать ее (рис. 78). Для этого существует специальный *вид* величины — ***результаты***.

Рис. 78

Рассмотрим пример алгоритма с результатом.

A 62

```
алг гипотенуза (арг вещ a, b, рез вещ c)
   дано a >= 0 и b >= 0 | длины катетов треугольника
   надо | c = длина гипотенузы этого треугольника
нач
|  c := sqrt (a**2 + b**2)
кон
```

Запись **рез вещ** с означает, что результат (**рез**) выполнения этого алгоритма — одна вещественная (**вещ**) величина, которая в алгоритме обозначена как *c*.

17.2. Выполнение алгоритма с результатами

В команде вызова вспомогательного алгоритма на месте его результатов указываются имена величин основного алгоритма, в которых должны оказаться вычисленные значения. Например, если нам надо в основном алгоритме "вычисление" найти гипотенузу x прямоугольного треугольника с катетами $p - 1$ и $p + 1$, то достаточно написать вызов вспомогательного алгоритма "гипотенуза" (А62):

```
алг вычисление
нач вещ p, x
|  ...
|  гипотенуза (p – 1, p + 1, x)
|  ...
кон
```

Пусть к моменту выполнения этого вызова величина *p* имеет значение 7 (рис. 79).

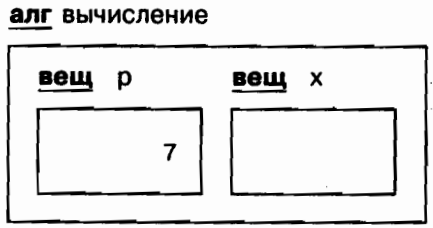


Рис. 79

Встретив вызов "гипотенуза (p – 1, p + 1, x)", компьютер вычислит значения $p - 1$ и $p + 1$ и передаст их алгоритму "гипотенуза" в качестве значений аргументов *a* и *b*. Значение *x* в этот момент не вычисляется, так как *x* соответствует *результату* вспомогательного алгоритма.

Затем начинается выполнение вспомогательного алгоритма "гипотенуза". Выделяется память алгоритма, в ней создаются ячейки для аргументов *a*, *b* и результата *c*, аргументы получают значения, переданные из основного алгоритма (рис. 80).

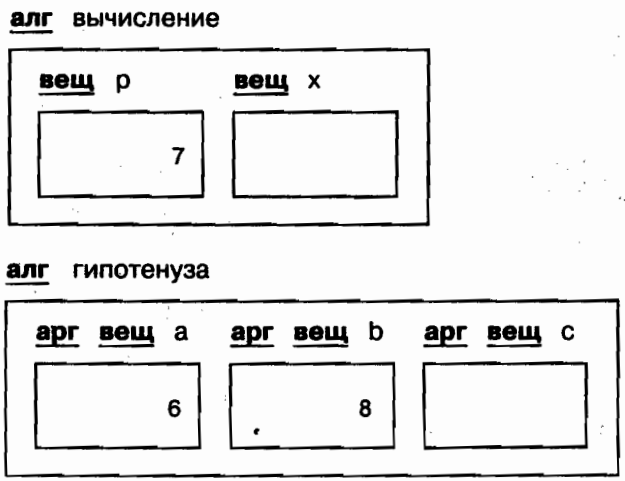


Рис. 80

В памяти компьютера при этом будет одновременно храниться информация, относящаяся и к алгоритму "вычисление", и к алгоритму "гипотенуза". Далее ЭВМ выполнит алгоритм "гипотенуза" и вычислит значение величины-результата *c* (рис. 81).

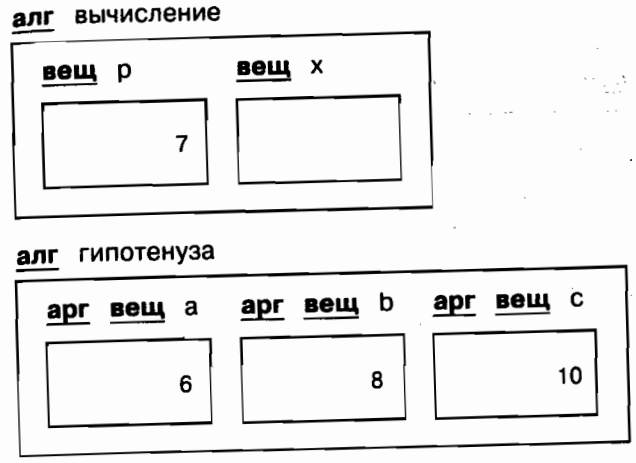


Рис. 81

Встретив строчку **кон**, компьютер передаст значение результата в основной алгоритм и завершит работу алгоритма "гипотенуза", освободив его память. В основном алгоритме значение результата записывается в ячейку величины, указанной в команде вызова, т. е. в *x* (рис. 82). После этого компьютер продолжает выполнять алгоритм "вычисление".

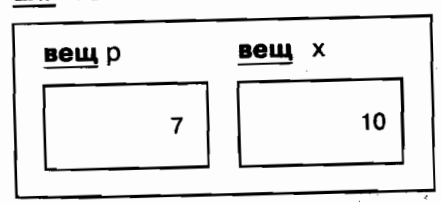


Рис. 82

17.3. Общие правила выполнения команды вызова вспомогательного алгоритма

1. Выполняя команду вызова вспомогательного алгоритма, компьютер вычисляет значения *аргументов* и передает их вспомогательному алгоритму в том порядке, в котором они записаны в команде вызова. После этого выполнение основного алгоритма приостанавливается.

2. Перед началом выполнения вспомогательного алгоритма компьютер отводит для него место в памяти. Аргументы получают значения, переданные из основного алгоритма.

3. В конце выполнения вспомогательного алгоритма его результаты передаются в основной алгоритм и заносятся в ячейки величин, указанных в команде вызова в качестве результатов.

4. После окончания вспомогательного алгоритма все, что с ним связано, стирается из памяти компьютера. (Если алгоритм вызывается еще раз, то все начинается сначала: компьютер снова отводит место в памяти, присваивает значения аргументам и т. д.)

5. Продолжается выполнение основного алгоритма.

17.4. Алгоритм с результатами при управлении *Роботом*

А 63

```
алг средний уровень радиации в коридоре (рез вещ r)
    дано | Робот в левой клетке коридора, уходящего вправо
    надо | r = средний уровень радиации в коридоре, Робот
         | вышел из коридора вправо
нач цел n, вещ S
|   n := 0; S := 0
|   нц пока снизу стена
|   |   n := n + 1; S := S + радиация
|   |   вправо
|   кц
|   утв | n = длина коридора, S = суммарная радиация
|   r := S/n
кон
```

17.5. Алгоритм Евклида

Рассмотрим такую задачу: найти наибольший общий делитель двух натуральных чисел. Один из способов решения этой задачи был предложен древнегреческим математиком Евклидом, жившим более двух тысяч лет назад. Суть метода Евклида в следующем: большее из двух чисел надо заменить разностью этих чисел. При этом все общие делители, а значит, и наибольший из них, сохраняются. Если проделать эту операцию несколько раз, то в конце концов числа станут равны друг другу и полученное значение будет искомым.

Запишем алгоритм Евклида на алгоритмическом языке.

А 64

```
алг Евклид (арг цел a, b, рез цел d)
    дано a > 0 и b > 0
    надо | d = НОД (a, b)
нач цел m, n | дополнительные величины нужны, чтобы не менять
|            | значения аргументов
|   m := a; n := b
|   нц пока m <> n
|   |   утв | инвариант цикла: НОД (m, n) = НОД (a, b)
|   |   если m > n
|   |   |   то m := m - n
|   |   |   иначе n := n - m
|   |   все
|   кц
|   утв m = n | НОД (m, n) = НОД (a, b)
|   d := m
кон
```

Докажем правильность этого алгоритма. Рассмотрим инвариант цикла. Ясно, что он справедлив при первом выполнении цикла — в этот момент значения m и n совпадают со значениями a и b.

Предположим, что при некотором выполнении цикла инвариант справедлив. Пусть в этот момент $m > n$. Тогда выполнится команда m := m – n и величина m получит новое значение. При этом все общие делители m и n остаются прежними, так как если m и n делятся на x, то $m - n$ также делится на x. Аналогично сохраняются все делители, если $n > m$.

Таким образом, инвариант цикла справедлив при первом выполнении и сохраняется при переходе от одного выполнения к следующему. Следовательно, он справедлив при любом выполнении цикла и после его завершения.

Условие, записанное после цикла, следует из инварианта цикла и условия окончания цикла ($m = n$). Из чего очевидно вытекает НОД $(a, b) = m$.

Осталось доказать, что выполнение цикла завершится. Для этого заметим, что при каждом выполнении цикла большее из чисел m и n уменьшается, но при этом они остаются положительными. Ясно, что это не может продолжаться бесконечно долго, следовательно, выполнение цикла когда-нибудь завершится.

17.6. Сумма цифр десятичного числа

Задача. Найти сумму цифр заданного десятичного числа.

Составим сначала заголовок алгоритма. В задаче есть одна входная величина — заданное число и одна выходная — сумма его цифр. Значит, в алгоритме будет один аргумент и один результат.

```
алг сумма цифр (арг цел n, рез цел s)                    A 65
    дано n > 0
    надо | s = сумма цифр числа n
    . . . . . .
```

Чтобы решить эту задачу, надо научить компьютер раскладывать число на цифры. Дело в том, что в памяти компьютера числа представляются совсем не так, как мы пишем их на бумаге, и привычное нам понятие «цифра» к величинам не применимо.

Для разделения числа на цифры надо понять, как устроена привычная нам десятичная запись числа. Последняя цифра числа — это остаток от его деления на 10, а все число без последней цифры — результат деления числа нацело на 10.

В алгоритмическом языке есть специальные операции для нахождения остатка и результата деления нацело (см. § 15). С их помощью алгоритм решения этой задачи можно записать так:

```
нач цел nn, c
|   nn := n  | значение аргумента нельзя изменять, поэтому нужна
|            | дополнительная промежуточная величина
|   s := 0
|   нц пока nn > 0
|   |   c:= mod (nn, 10)
|   |   s := s + c
|   |   nn := div (nn, 10)
|   кц
кон
```

17.7. Исполнение алгоритмов

Чтобы лучше понять работу того или иного алгоритма, полезно проследить ее по шагам без использования компьютера. Это легко сделать, если алгоритм управляет каким-то исполнителем — можно моделировать действия исполнителя и следить за результатом.

Если алгоритм работает с величинами, для пошаговой проверки удобно пользоваться специальной формой записи, которую мы разберем на примере. Пусть выполнен вызов "сумма цифр (197, x)". Исполнение тела алгоритма "сумма цифр" можно в этом случае представить так:

```
nn := n                    197
s := 0                       0
нц пока nn > 0            да      да      да      нет
|   c := mod (nn, 10)       7       9       1
|   s := s + c               7      16      17
|   nn := div (nn, 10)      19       1       0
кц
```

В этой записи правее каждой команды алгоритма записаны последовательно результаты ее выполнения. Например, первая команда nn := n выполняется один раз, и величина nn получает значение 197. Команда c := mod(nn, 10) выполняется три раза, при ее выполнении величина c последовательно получает значения 7, 9 и 1, т. е. значения цифр исходного числа в обратном порядке.

ЗАДАЧИ И УПРАЖНЕНИЯ

1. Составьте алгоритмы со следующими заголовками:

а) **алг** расстояние до стены вправо (**рез цел** n)
```
    дано | правее Робота есть стена
    надо | положение Робота не изменилось, n = число
         | шагов, которое надо сделать Роботу, чтобы
         | подойти к стене вплотную (т. е. число клеток
         | между Роботом и стеной)
```

б) **алг** расстояния вправо и влево (**рез цел** n1, n2)
```
    дано | правее и левее Робота есть стены
    надо | положение Робота не изменилось, n1 = расстояние
         | вправо до стены, n2 = расстояние влево до стены
```

в) **алг** к ближайшей стене вправо или влево
```
    дано | правее и левее Робота есть стены
    надо | Робот подошел к ближайшей из этих стен
```

г) **алг** подсчет числа опасных клеток (**рез цел** n)
```
    дано | Робот на клетку левее тупика, уходящего вправо
    надо | положение Робота не изменилось, n = число кле-
         | ток, в которых и температура, и уровень радиации
         | более чем вдвое выше средних значений в тупике
```

2. Составьте алгоритм, который вычисляет:
   а) количество максимальных среди трех чисел *a, b, c*;
   б) количество разных среди трех чисел *a, b, c*;
   в) среднее по величине среди трех чисел *a, b, c*;
   г) расстояние от точки на числовой оси с координатой *x* до ближайшей точки отрезка [0, 1];
   д) наименьший делитель целого числа $n > 0$, отличный от 1;
   е) последнюю цифру в десятичной записи целого числа $n > 0$;
   ж) количество цифр в десятичной записи целого числа $n > 0$;
   з) старшую цифру в десятичной записи целого числа $n > 0$.

3. Составьте схему пошагового выполнения алгоритма Евклида (А64) при $a = 14$, $b = 21$.

18 Команды ввода/вывода информации

18.1. Необходимость ввода и вывода информации

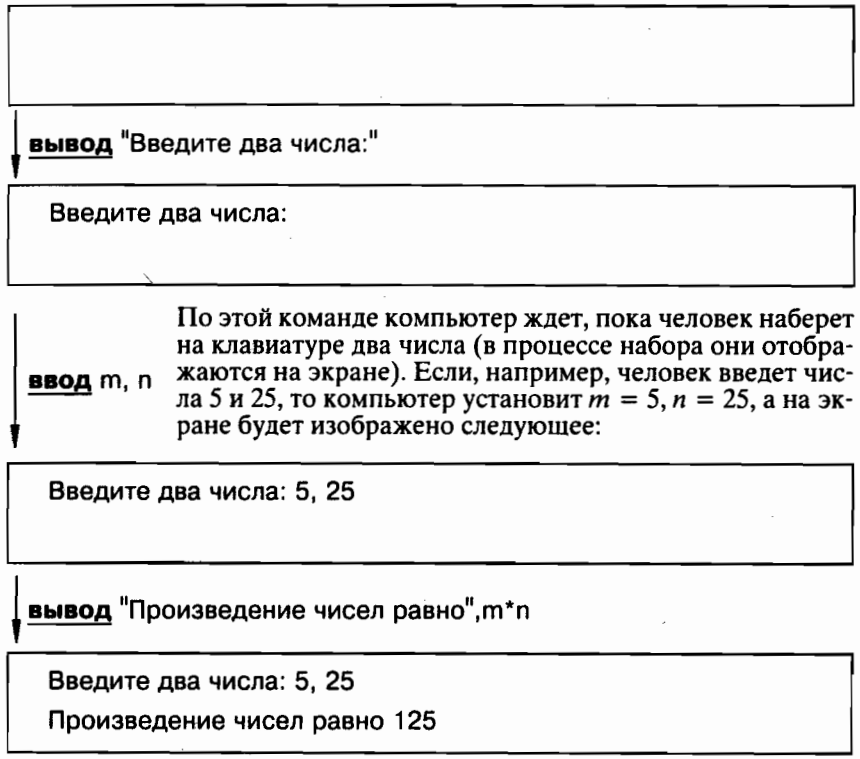
До сих пор, составляя алгоритмы, мы считали, что компьютер выполняет их автоматически, без какого-либо взаимодействия с человеком. Часто, однако, требуется организовать обмен информацией (диалог) между человеком и компьютером в процессе выполнения алгоритма. Для этого в алгоритмическом языке есть специальные команды ***вывода*** информации из памяти компьютера на экран и ***ввода*** информации с клавиатуры (от человека) в память компьютера. Поскольку в алгоритмическом языке для запоминания информации используются величины, то в командах ввода/вывода указываются имена величин, значения которых надо вывести (показать на экране) или ввести (запомнить в памяти компьютера).

18.2. Простейший пример алгоритма с командами ввода/вывода

А 66

```
алг произведение
нач цел m, n
| вывод "Введите два числа:"
| ввод m, n
| вывод "Произведение чисел равно", m*n
кон
```


При выполнении этого алгоритма экран будет меняться следующим образом:



18.3. Работа команд ввод и вывод

В команде **ввод** через запятую указываются имена величин. При выполнении команды компьютер ждет, пока человек введет соответствующие значения, и по окончании ввода присваивает эти значения указанным величинам. Признаком конца ввода служит нажатие на клавишу Enter. При вводе нескольких чисел они отделяются друг от друга запятыми или пробелами.

В команде **вывод** можно через запятую указать тексты, имена величин и выражения. При выполнении команды компьютер вычислит значения выражений, подставит значения величин и изобразит на экране то, что получится (если информация не уместится в одной строке, она будет перенесена на следующую).

Перейти на новую строку можно и до окончания текущей, написав в команде вывода служебное слово **нс** (новая строка).

Если в процессе вывода информация заполнит весь экран, то при переходе на следующую строку весь экран сдвинется вверх, вытеснив верхнюю строку «за экран». Таким образом, при продолжении вывода изображение побежит по экрану снизу вверх и в конце на экране останутся лишь последние из выведенных строк.

18.4. Нахождение среднего арифметического

Задача. Ввести последовательность чисел и вычислить их среднее арифметическое.

Ввод последовательности означает, что мы заранее не знаем, сколько чисел потребуется ввести, поэтому сначала запросим их количество. Затем будем по одному вводить сами числа, подсчитывая их сумму. При этом хранить все введенные числа не обязательно, поэтому для них вполне достаточно одной величины.

А 67

```
алг среднее арифметическое
нач цел n | количество чисел
|   вещ x      |очередное число
|   вещ S      |сумма чисел
|   вывод "Сколько будет чисел?"; ввод n
|   S := 0
|   нц n раз
|   |   вывод "Введите число:"; ввод x
|   |   S := S + x
|   кц
|   вывод "Среднее арифметическое=", S/n
кон
```

18.5. Диалоговые алгоритмы

В рассмотренных алгоритмах команды ввода/вывода использовались для ввода начальных данных и вывода результатов вычислений, т. е. по схеме «ввод информации — обработка — вывод». При такой схеме обработка информации производится автоматически, без участия человека.

Команды ввода/вывода могут быть применены и для организации совместной работы человека и компьютера, когда решения принимает человек, а рутинную работу выполняет компьютер. Такие алгоритмы называются ***диалоговыми***.

ЗАДАЧИ И УПРАЖНЕНИЯ

1. Составьте алгоритм, который:
   а) запрашивает два числа и выводит их полусумму;
   б) запрашивает пять целых чисел и выводит квадраты этих чисел;
   в) запрашивает четыре вещественных числа *a, b, c, d* и с помощью *Чертежника* рисует отрезок от точки (*a, b*) до точки (*c, d*);
   г) запрашивает три вещественных числа *a, b, r* и с помощью *Чертежника* приближенно рисует окружность радиуса *r* с центром в точке (*a, b*);
   д) запрашивает целое число *n* в диапазоне от 1 до 100 и сообщает, простое оно или нет.
2. Составьте алгоритм, который вводит последовательность чисел и сообщает:
   а) их произведение;
   б) наибольшее из всех чисел последовательности;
   в) количество отрицательных чисел в последовательности;
   г) количество повторений первого числа в последовательности.
3. Выполните упражнение 3 из § 17, используя ввод и вывод вместо задания аргументов и результатов.

19 Алгоритмы-функции

19.1. Пример алгоритма-функции

Задача. Ребра прямоугольного параллелепипеда имеют размеры *a, b* и *c*. Определить периметр треугольника, образованного диагоналями граней (рис. 83).

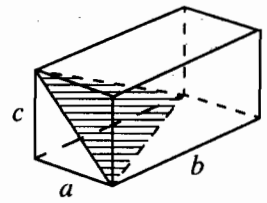


Рис. 83

Ясно, что надо вычислить длины трех диагоналей и сложить их. Воспользовавшись вспомогательным алгоритмом "гипотенуза" (А62), получаем такой алгоритм:

А 68

```
алг треугольник на гранях (арг вещ a, b, c, рез вещ p)
   дано a > 0 и b > 0 и c > 0
   надо | p — периметр треугольника (рис. 83)
нач вещ x, y, z
|   гипотенуза (a, b, x)
|   гипотенуза (b, c, y)
|   гипотенуза (c, a, z)
|   p := x + y + z
кон
```

Полученный алгоритм выглядит несколько громоздко. Простые слова «сумма трех гипотенуз» превратились в нем в четыре команды! Возникает желание записать эти действия несколько короче и одновременно указать, что надо вычислить гипотенузы и сложить их.

Для этого можно воспользоваться специальным средством алгоритмического языка — ***алгоритмом-функцией***. При этом алгоритмы "треугольник на гранях" и "гипотенуза" будут выглядеть так:

А 69

```
алг треугольник на гранях (арг вещ a, b, c, рез вещ p)
    дано a > 0 и b > 0 и c > 0
    надо | p — периметр треугольника (рис. 83)
нач
|   p := гипотенуза (a, b) + гипотенуза (b, c) + гипотенуза (c, a)
кон
```

А 70

```
алг вещ гипотенуза (арг вещ a, b)
    дано a >= 0 и b >= 0 | длины катетов треугольника
    надо | знач = длина гипотенузы этого треугольника
нач
|   знач := sqrt (a**2 + b**2)
кон
```

Здесь "гипотенуза" — алгоритм-функция, а "треугольник на гранях" — обычный алгоритм, который *обращается* к функции.

19.2. Отличие функций от обычных алгоритмов

Алгоритм-функция — это не какой-то особый вид алгоритмов, а просто другой способ записи. Он отличается от обычного алгоритма формой записи заголовка и использованием специального служебного слова **знач**.

Алгоритм-функция обычно имеет единственный результат. Этот результат не включается в список параметров алгоритма — там записываются только аргументы. Вместо этого после слова **алг** перед именем алгоритма вставляется указатель типа этого результата. Например, запись "**алг вещ** гипотенуза (**арг вещ** a, b)" означает, что "гипотенуза" — это алгоритм-функция, имеющий два вещественных аргумента и один вещественный результат.

Тело алгоритма-функции записывается точно так же, как и тело обычного алгоритма, но вместо имени результата используется специальное служебное слово **знач**.

Вызов алгоритма-функции может вставляться в любое арифметическое выражение. При вычислении выражения на место вызова подставляется вычисленный результат функции.

19.3. Выполнение алгоритма-функции

Пусть в некотором алгоритме встречается команда

```
y := гипотенуза (3, 4) + 6
```

Рассмотрим работу компьютера при ее выполнении. Запись "гипотенуза (3, 4)" означает вызов алгоритма-функции "гипотенуза" с аргументами 3 и 4. Значения аргументов, как обычно, передаются во вспомогательный алгоритм.

Компьютер выделяет память для алгоритма "гипотенуза" и его аргументов. Кроме того, выделяется память для результата функции. Внутри функции результат всегда имеет имя **знач**, а его тип указывается в заголовке функции перед ее именем. После подстановки значений параметров память алгоритма "гипотенуза" выглядит, как на рисунке 84.

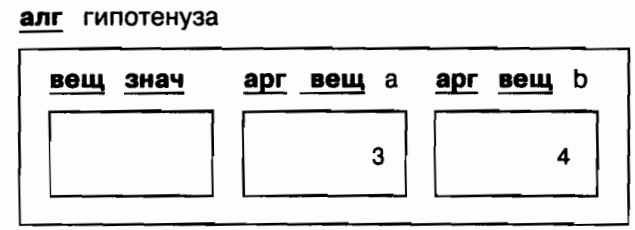


Рис. 84

Тело алгоритма "гипотенуза" состоит из одной команды:

```
знач := sqrt (a**2 + b**2)
```

В соответствии с правилами алгоритмического языка компьютер заменит в правой части имена аргументов на их зна-

чения, вычислит выражение sqrt (3**2 + 4**2) = sqrt (25) = 5 и присвоит полученное значение величине **знач** (рис. 85).

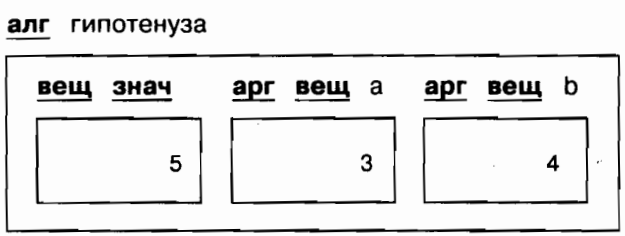


Рис. 85

На этом выполнение алгоритма-функции "гипотенуза" закончится. Встретив строку **кон**, компьютер перенесет полученное значение величины **знач** — число 5 — в основной алгоритм и сотрет из памяти прямоугольник "**алг** гипотенуза" со всем его содержимым.

В основном алгоритме результат подставляется в выражение на место вызова функции.

Далее компьютер вычислит выражение 5 + 6 = 11 и присвоит полученное значение величине *y*. На этом выполнение команды присваивания "y := гипотенуза (3, 4) + 6" закончится.

19.4. Построение графика произвольной функции

Если заменить в алгоритме рисования параболы (А60) *x***2 на *f*(*x*), можно получить алгоритм рисования графика произвольной функции *f*, заданной алгоритмом-функцией

алг вещ f (**арг вещ** x):

А 71

```
алг график (арг вещ a, b, арг цел n)
    дано n > 0  | перо Чертежника поднято
    надо | нарисован график функции y = f(x) на участке от a до b
         | в виде ломаной из n звеньев;
         | перо в точке (b, f(b)) и поднято
нач вещ x, d
|   x := a; d := (b – a)/n
|   сместиться в точку (a, f(a))
|   опустить перо
|   нц n раз
|   |   x := x + d
|   |   сместиться в точку (x, f(x))
|   кц
|   поднять перо
кон
```

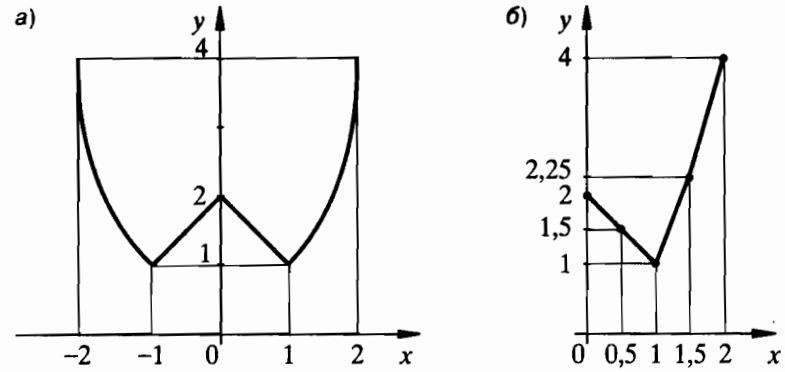



Рис. 86. а) График функции; б) ломаная из 4 звеньев

Например, для задания функции, изображенной на рисунке 86 *а*, можно составить такой алгоритм:

А 72

```
алг вещ f (арг вещ x)
нач
|   если –1 <= x <= 1                |          2 – |x|, если |x| <= 1
|   |   то знач := 2 – abs (x)       |  f (x) =
|   |   иначе знач :=x**2            |          x**2, если |x| > 1
|   все
кон
```

В этом случае при выполнении вызова "график (0, 2, 4)" компьютер 5 раз обратится к вспомогательному алгоритму *f* для вычисления значений функции *f* в точках 0,0; 0,5; 1,0; 1,5; 2,0 и, используя эти значения, выдаст *Чертежнику* 7 команд:

```
сместиться в точку (0, 2)
опустить перо
сместиться в точку (0.5, 1.5)
сместиться в точку (1.0, 1.0)
сместиться в точку (1.5, 2.25)
сместиться в точку (2.0, 4.0)
поднять перо
```

В результате будет нарисована ломаная, изображенная на рисунке 86, *б*. Поскольку в вызове указано, что ломаная должна иметь всего четыре звена, она значительно отличается от графика функции.

ЗАДАЧИ И УПРАЖНЕНИЯ

1. Составьте алгоритмы-функции со следующими заголовками:

а) **алг цел** число закрашенных клеток на север (**арг цел** n)
дано n >= 0 | выше Робота стен нет
надо | **знач** = число закрашенных клеток среди n клеток
| выше Робота, Робот в исходном положении

б) **алг цел** расстояние до стены вниз
дано | где-то ниже Робота есть стена
надо | положение Робота не изменилось, **знач** = число
| шагов, которое надо сделать Роботу, чтобы
| подойти к стене вплотную (т. е. число клеток
| между Роботом и стеной)

в) **алг цел** число выходов из клетки
надо | положение Робота не изменилось, **знач** = число
| сторон клетки, не перегороженных стенами

2. Выполните упражнение 3 из § 17, используя функции вместо обычных алгоритмов с результатами.

20 Логические величины

20.1. Тип величины

Как мы уже знаем, одна из важнейших характеристик любой величины — ее ***тип***, который определяет: 1) какие значения может принимать величина в процессе выполнения алгоритма; 2) в каких выражениях и как эту величину можно использовать.

До сих пор мы имели дело только с числовыми величинами типов **цел** и **вещ**. Значениями числовых величин бывают числа (целые для типа **цел**, вещественные — для **вещ**), которые можно использовать в арифметических выражениях и выполнять над ними различные операции (табл. 12). Кроме того, числовые величины можно сравнивать, получая при этом ***логические*** значения **да** или **нет**.

Мы уже не раз пользовались логическими значениями, которые получаются в результате проверки различных условий. В этом параграфе мы рассмотрим применение логических величин более подробно.

20.2. Логические величины, выражения и присваивания

Существует всего два логических значения. В алгоритмическом языке они обозначаются служебными словами **да** и **нет**. Чаще всего логические значения появляются в результате проверки условий. Проверив условие, компьютер получает некоторое логическое значение, в соответствии с которым принимает решение о дальнейшем выполнении алгоритма (например, выполнять или нет тело цикла **пока**). Само логическое значение после этой проверки не сохраняется в памяти компьютера.

Иногда возникает необходимость запомнить логическое значение, чтобы воспользоваться им позже. Для этого в алгоритмическом языке служат величины логического типа, или ***логические величины***.

Общие правила использования логических величин в алгоритме такие же, как у числовых величин. Логические величины необходимо описывать (для этого предназначено служебное слово **лог**), они могут быть аргументами, результатами, промежуточными величинами, значениями алгоритмов-функций.

Логическая величина принимает только одно из двух ***логических значений*** — либо **да**, либо **нет** (либо она не определена). Логические величины используются в ***логических выражениях***, над ними можно совершать операции **и**, **или**, **не**. Условия, которые мы записывали в командах **пока**, **если** и **выбор**, есть на самом деле не что иное, как логические выражения.

Для установки или изменения значения величины удобна команда ***логического присваивания***:

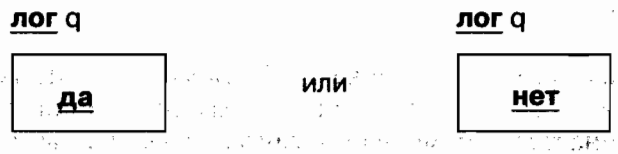
имя величины := условие (логическое выражение)

Например, для величины **лог** q можно написать

q := 0 < t < 1 **и** снизу стена

При выполнении команды логического присваивания компьютер сначала проверяет записанное справа условие (говорят также *вычисляет логическое выражение*). Условие проверяется так же, как проверяются условия в командах **пока** и **если**. В результате проверки получается либо **да** (условие соблюдает-

ся), либо **нет** (условие не соблюдается). Полученное значение (т. е. **да** или **нет**) записывается в память логической величины:



Логические величины можно использовать в логических выражениях (т. е. в условиях) наравне со сравнениями и вызовами команд-вопросов, например:

если q **то** ...
нц пока q **или** клетка закрашена ...

При проверке таких условий компьютер просто подставляет вместо имени логической величины ее значение.

20.3. Пример алгоритма с логическими величинами и выражениями

A 73

```
алг обработка тупика
   дано | Робот левее горизонтального тупика (рис. 87)
   надо | Робот в исходном положении;
        | если в тупике была хоть одна закрашенная клетка,
        | то закрашен весь тупик
нач лог q
|  q := нет | закрашенные клетки пока не обнаружены
|  нц пока справа свободно
|  |  вправо
|  |  q := q или клетка закрашена
|  кц
|  утв | Робот в самой правой клетке тупика, q = "в тупике есть
|      | закрашенная клетка"
|  нц пока снизу стена | т. е. пока в тупике
|  |  если q | т. е. если была хоть одна закрашенная клетка
|  |  | то закрасить
|  |  все
|  |  влево
|  кц
кон
```

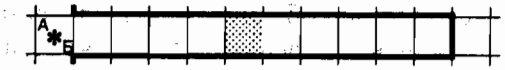



Рис. 87

В данном алгоритме при движении *Робота* вправо по тупику информация о том, встретил ли он хотя бы одну закрашенную клетку, запоминается с помощью логической величины *q*. На обратном пути сохраненная информация используется. Это пример так называемой отложенной проверки: значение условия запоминается, а проверяется позднее, когда в этом возникает необходимость.

20.4. Пример логического алгоритма-функции

A 74

```
алг лог перекресток
   надо | если из клетки можно уйти в любом направлении,
        | то знач = да. В противном случае знач = нет
нач
|  знач := справа свободно и слева свободно и сверху свободно
|          и снизу свободно
кон
```

Используя этот алгоритм-функцию как вспомогательный, можно составить, например, следующий алгоритм:

A 75

```
алг лог вправо до перекрестка
   дано | Робот в горизонтальном коридоре, левее перекрестка
   надо | Робот на перекрестке
нач
|  нц пока не перекресток
|  |  вправо
|  кц
кон
```

При проверке условия "**не** перекресток" в цикле **пока** компьютер будет вызывать вспомогательный алгоритм-функцию А74 и подставлять вычисленное значение вместо имени функции "перекресток".

20.5. Логический алгоритм-функция в методе последовательного уточнения

Пусть известно, что *Робот* находится в левой клетке горизонтального коридора. Из некоторых клеток коридора вверх отходят тупики. Требуется закрасить клетки коридора, от которых вверх отходят тупики длиной больше трех клеток (рис. 88).

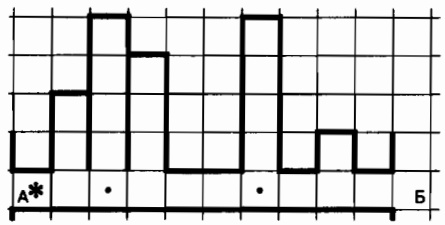


Рис. 88

Для решения этой задачи мы можем провести *Робота* по коридору слева направо, в каждой клетке коридора проанализировать, не выходит ли из нее длинный тупик (т. е. тупик, длиной больше трех клеток), и если выходит — то закрасить клетку. Используя метод последовательного уточнения, этот план можно почти дословно записать так:

А 76

```
алг разметка коридора
    дано | Робот в левой клетке горизонтального коридора, из
         | некоторых клеток коридора вверх отходят тупики
    надо | Робот вышел из коридора вправо, клетки коридора
         | напротив тупиков длиной больше 3 клеток закрашены
нач
| нц пока снизу стена | т. е. пока Робот в коридоре
| | если сверху длинный тупик
| | то закрасить
| | все
| | вправо
| кц
кон
```

Здесь использован вызов еще не написанного вспомогательного логического алгоритма-функции "сверху длинный тупик". Теперь, в соответствии с методом последовательного уточнения, составим этот алгоритм-функцию:

А 77

```
алг лог сверху длинный тупик
    надо | знач = да, если вверх отходит тупик длины > 3 клеток,
         | Робот в исходном положении
нач цел n
| n := 0
| нц пока сверху свободно
| | вверх; n := n + 1
| кц
| вниз на (n)
| знач := (n > 3)
кон
```

В результате сравнения $n > 3$ компьютер получает либо **да**, либо **нет**. Это логическое значение присваивается величине **знач**.

ЗАДАЧИ И УПРАЖНЕНИЯ

1. Перед выполнением $a = 1$, $b = 2$, $c = 3$, $q =$ **нет**. Определите значение величины q после выполнения каждой из следующих команд присваивания:

а) q := **да**
б) q := **не** q
в) q := a < b < c **и** q
г) q := q **или** (a + b = c)
д) q := 0 > 1
е) q := 1 > 0

2. Составьте алгоритмы со следующими заголовками:

а) **алг лог** клетка опасна (**арг вещ** a)
```
    дано | на поле Робота стен нет
    надо | знач = да, если суммарный уровень радиации
         | в клетке с Роботом и в ее 8 соседях превышает a;
         | положение Робота не изменилось
```

б) **алг лог** четно (**арг цел** n)
```
    надо | знач = да, если n четно
```

3. Что произойдет при попытке выполнить алгоритм А76 в обстановке, когда от коридора вверх отходит другой коридор (не тупик)?

4. Измените алгоритм А77 так, чтобы при выполнении алгоритма А76 закрашивались клетки напротив любых ответвлений вверх (не обязательно тупиков) длиной больше трех клеток.

5. Составьте алгоритм со следующим заголовком:

алг число проходов (**рез цел** n)
дано | Робот в левом нижнем углу прямоугольника, огороженного стенами. В нижней стене прямоугольника есть несколько проходов разной ширины
надо | Робот в правом нижнем углу прямоугольника, n = число проходов в нижней стене

21 Табличные величины и работа с ними

21.1. Табличные величины позволяют работать с большими объемами информации

Секрет могущества компьютера — высокая скорость и большая память. Мы уже научились использовать скорость работы компьютера: команды циклов позволяют составлять короткие алгоритмы, при выполнении которых компьютер быстро совершает очень длинные последовательности действий. До сих пор, однако, во всех наших алгоритмах объем информации, хранимый в памяти компьютера в процессе выполнения алгоритма, был очень невелик.

А как быть, если нужно работать с сотней, тысячей или сотней тысяч чисел? Понятно, что невозможно описывать в алгоритме такое количество отдельных величин. А ведь надо еще и указать обработку этой информации, и если все числа обрабатываются одинаково, то хотелось бы иметь возможность задать единый алгоритм для всех, а не писать его отдельно для каждого.

Для этих целей применяется специальный способ организации информации — ***табличные величины*** (или просто ***таблицы***).

21.2. Простые и составные величины

Типы величин в алгоритмическом языке бывают ***простые*** и ***составные***. Целый, вещественный, логический типы относятся к простым. Величины этого типа занимают одну ячейку памяти и в каждый момент времени имеют единственное значение.

Табличные величины относятся к составным. Каждая таблица состоит из величин какого-то типа, одинакового для всех величин, входящих в таблицу. Этот общий тип называется ***базовым*** типом таблицы. В качестве базового может служить любой из уже известных нам типов: целый, вещественный, логический. Чаще всего применяются числовые таблицы, состоящие из целых или вещественных величин.

Величины, составляющие таблицу, называются ее ***элементами***. Количество элементов называется ***размером*** таблицы.

21.3. Описание таблицы и размещение ее в памяти компьютера

В памяти компьютера таблица занимает несколько ячеек памяти — по одной для каждого элемента. Все элементы таблицы нумеруются. Номера элементов называются ***индексами***.

Как и любую другую величину, таблицу в алгоритме необходимо перед использованием описать. Описание таблицы, кроме указания, что это табличная величина, включает в себя имя, размер и базовый тип таблицы. Например, описание **цел таб** k[1:5] задает таблицу с именем *k*, состоящую из пяти элементов целого типа. В соответствии с этим описанием для таблицы *k* выделяется пять ячеек памяти:

цел таб k[1:5]

индексы элементов:	1	2	3	4	5

В процессе работы алгоритма таблица заполняется информацией. Например, таблица *k* может в какой-то момент выглядеть так:

цел таб k[1:5]

значения элементов:	9	4	0	–17	123
индексы элементов:	1	2	3	4	5

Здесь значение таблицы *k* — это пять целых чисел: 9, 4, 0, –17 и 123.

21.4. Работа с элементами таблиц

Для обращения к конкретному элементу надо указать имя таблицы и в квадратных скобках после него — номер нужного элемента.

Рассмотрим несколько примеров.

цел таб k[1:5]

9	4	0	−17	123
1	2	3	4	5

k[3] := k[2] + k[4]

При выполнении этой команды компьютер подставит вместо $k[2]$ и $k[4]$ значения 2-го и 4-го элементов таблицы k, т. е. числа 4 и −17, сложит их и присвоит полученное значение 3-му элементу:

цел таб k[1:5]

9	4	−13	−17	123
1	2	3	4	5

k[k[2] + 1] := 0

При выполнении этой команды компьютер вычислит выражение $k[2] + 1 = 4 + 1 = 5$ и выполнит команду присваивания $k[5] := 0$.

цел таб k[1:5]

9	4	−13	−17	0
1	2	3	4	5

21.5. Пример алгоритма работы с таблицей

A 78

```
алг очистка (рез вещ таб A[1:100])
   надо | все элементы таблицы A равны 0
нач цел i
|  i := 1                          | *
|  нц пока i <= 100                | *
|  |  A[i] := 0
|  |  i := i + 1                   | *
|  кц
кон
```


21.6. Цикл для

Обратите внимание на строчки, отмеченные звездочками в алгоритме А78. Они обеспечивают перебор всех элементов таблицы A. Величина i последовательно принимает значения 1, 2, ... и т. д. до 100, и при каждом выполнении цикла обрабатывается i-й элемент таблицы A.

Такая схема поэлементной обработки таблиц очень часто встречается в различных задачах. Для упрощения записи и понимания алгоритмов, содержащих подобную обработку, в алгоритмическом языке существует специальный вид цикла — цикл **для**. С использованием цикла **для** алгоритм А78 можно переписать так:

A 79

```
алг очистка (рез вещ таб A[1:100])
   надо | все элементы таблицы A равны 0
нач цел i
|  нц для i от 1 до 100
|  |  A[i] := 0
|  кц
кон
```

21.7. Общий вид цикла для

Общий вид цикла **для** таков:

```
нц для i от i1 до i2
|  тело цикла (последовательность команд)
кц
```

Здесь i — имя величины целого типа (она называется управляющей переменной цикла), $i1$, $i2$ — произвольные целые числа или выражения с целыми значениями. Тело цикла последовательно выполняется для $i = i1$, $i = i1 + 1$, $i1 + 2$, ... $i = i2$.

Правила алгоритмического языка допускают задание любых целых $i1$, $i2$. В частности, $i2$ может быть меньше $i1$. Этот случай не считается ошибочным — просто тело цикла не будет выполнено ни разу, а компьютер сразу перейдет к выполнению команд, записанных после **кц**.

21.8. Таблицы переменного размера

В алгоритмах А78 и А79 размер таблицы был задан в заголовке алгоритма. Часто это бывает неудобно: для решения той же самой задачи для таблицы с другим количеством элементов надо составлять новый алгоритм.

С подобной проблемой мы уже встречались (§ 14), тогда мы разрешили ее, введя в алгоритмы аргументы. Аналогичный прием применяется и при обработке таблиц. В заголовке описывается таблица переменного размера, например A[1:n]. При этом значение *n* должно быть известно, так как компьютеру необходимо знать, сколько ячеек памяти выделить для таблицы. Чаще всего размер таблицы задается как ***аргумент*** алгоритма, в котором применяется таблица переменного размера.

Например, алгоритм "очистка" с использованием таблицы переменного размера можно переписать так:

А 80

```
алг очистка (арг цел n, рез вещ таб A[1:n])
   надо | все элементы таблицы A равны 0
нач цел i
|  нц для i от 1 до n
|  |  A[i] := 0
|  кц
кон
```

В дальнейшем мы будем составлять все алгоритмы работы с табличными величинами для таблиц переменного размера.

21.9. Задачи обработки таблиц

При решении на компьютере реальных задач приходится обрабатывать большие объемы однородной информации. Обычно эта информация представляется в форме таблиц. Для обработки табличной информации применяются различные алгоритмы, в том числе довольно сложные.

Однако чаще всего обработка табличных величин сводится к применению нескольких (иногда многих) сравнительно простых алгоритмов. Поэтому, научившись составлять алгоритмы решения некоторых наиболее распространенных задач, можно успешно применять их и в более сложных задачах.

Наиболее часто встречаются следующие виды задач:

1) ***Задачи заполнения*** — необходимо заполнить таблицу информацией в соответствии с заданным правилом.

2) ***Задачи анализа*** — дана уже заполненная таблица и требуется найти какую-то ее характеристику (например, сумму всех элементов).

3) ***Задачи поиска*** — надо определить, где в заполненной таблице находится элемент с заданными свойствами.

4) ***Задачи перестановки*** — следует переставить местами элементы таблицы в соответствии с заданным правилом.

21.10. Задачи заполнения

В задачах заполнения таблица выступает как *результат* алгоритма. Обычно в этих задачах нужно перебрать все элементы (для чего удобно использовать цикл **для**) и записать в каждый из них нужную информацию.

Пример 1. Самый простой пример — очистка таблицы, т. е. заполнение ее нулями, был разобран ранее (А80).

Пример 2. Заполним таблицу квадратами натуральных чисел. Надо получить A[1] = 1, A[2] = 4, A[3] = 9 и т. д.

А 81

```
алг квадраты (арг цел n, рез вещ таб A[1:n])
   надо | таблица A заполнена квадратами натуральных чисел
нач цел i
|  нц для i от 1 до n
|  |  A[i] := i**2
|  кц
кон
```

Пример 3. В математике хорошо известна последовательность Фибоначчи. Она начинается так: 1, 1, 2, 3, 5, 8, 13, 21... . Первые два члена последовательности равны 1, а каждый следующий равен сумме двух предыдущих. Составим алгоритм, заполняющий таблицу элементами последовательности Фибоначчи.

А 82

```
алг Фибоначчи (арг цел n, рез вещ таб A[1:n])
    дано n > 1
    надо | таблица A заполнена элементами последовательности
         | Фибоначчи
нач цел i
|   A[1] := 1; A[2] := 1
|   нц для i от 3 до n
|   |   A[i] := A[i – 2] + A[i – 1]
|   кц
кон
```

21.11. Задачи анализа

В задачах анализа нужно найти какую-то характеристику заданной таблицы. При этом таблица выступает как **аргумент** алгоритма. Обычно в этих задачах нужно перебрать все элементы (для этого удобно использовать цикл **для**) и выполнить с каждым из них какие-то действия.

21.12. Однопроходные алгоритмы

При решении многих задач анализа можно применить специальный метод, который называется ***построение однопроходных алгоритмов***.

Суть его в следующем. Сначала следует найти требуемую характеристику для таблицы из одного элемента. Обычно это бывает нетрудно. Затем надо понять, как изменяется эта характеристика при добавлении к произвольной таблице одного элемента, и выполнить эти изменения для всех элементов, начиная со второго.

Задача 1. Найти сумму всех элементов таблицы. Ясно, что для таблицы из одного элемента сумма равна этому элементу. Ясно также, что при добавлении к произвольной таблице одного элемента сумма увеличивается на значение добавленного элемента. Получаем такой алгоритм:

А 83

```
алг сумма (арг цел n, вещ таб A[1:n], рез вещ S)
    надо | S = сумма элементов таблицы A
нач цел i
|   S := A[1]
|   нц для i от 2 до n
|   |   S := S + A[i]
|   кц
кон
```

Задача 2. Найти максимальное значение, встречающееся в числовой таблице. Ясно, что для таблицы из одного элемента максимум равен этому элементу. При добавлении одного элемента к произвольной таблице максимум изменяется так: если новый элемент больше уже известного максимума, этот элемент становится новым максимумом, в противном случае максимум не изменяется.

А 84

```
алг максимум (арг цел n, вещ таб A[1:n], рез вещ макс)
    надо | макс = максимальное число в таблице A
нач цел i
|   макс := A[1]
|   нц для i от 2 до n
|   |   если A[i] > макс
|   |   |   то макс := A[i]
|   |   все
|   кц
кон
```

Применяя метод однопроходных алгоритмов, иногда бывает удобно рассмотреть сначала таблицу не из одного элемента, а пустую, не содержащую элементов вообще. Например, для задачи 1 о нахождении суммы элементов при этом получается такой алгоритм:

А 85

```
алг сумма (арг цел n, вещ таб A[1:n], рез вещ S)
    надо | S = сумма элементов таблицы A
нач цел i
|   S := 0
|   нц для i от 1 до n
|   |   S := S + A[i]
|   кц
кон
```

Задача 3. Найти в таблице количество положительных чисел. В пустой таблице нет никаких чисел, в том числе и положительных — их количество равно нулю. При добавлении одного элемента количество положительных увеличивается на 1, если новый элемент положителен, и не изменяется в противном случае.

А 86

```
алг число положительных (арг цел n, вещ таб A[1:n], рез цел k)
   надо | k = число элементов таблицы A, больших 0
нач цел i
|  k := 0
|  нц для i от 1 до n
|  |  если A[i] > 0
|  |  | то k := k + 1
|  |  все
|  кц
кон
```

Метод однопроходных алгоритмов позволяет решать и более сложные задачи, алгоритм для которых не всегда удается найти другим способом.

Задача 4. Сколько раз встречается в таблице ее максимальный элемент? Эту задачу можно легко решить в два прохода: сначала найти значение максимального элемента, а затем подсчитать их количество.

Но, рассуждая по правилам построения однопроходных алгоритмов, можно найти решение, позволяющее обойтись одним просмотром таблицы.

Для таблицы из одного элемента этот элемент является максимальным и количество таких элементов, естественно, равно 1. При добавлении к произвольной таблице одного элемента возможны следующие случаи:

а) новый элемент меньше имеющегося максимума — результат не изменяется;

б) новый элемент равен имеющемуся максимуму — результат увеличивается на 1;

в) новый элемент больше имеющегося максимума — устанавливается новый максимум, а их количество становится равным 1.

Осталось только оформить эти рассуждения в виде алгоритма:

А 87

```
алг число максимумов (арг цел n, вещ таб A[1:n], рез цел k)
   надо | k = число максимальных элементов в таблице A
нач цел i, вещ макс
|  макс := A[1]; k := 1
|  нц для i от 2 до n
|  |  выбор
|  |  | при A[i] = макс: k := k + 1
|  |  | при A[i] > макс: k := 1; макс := A[i]
|  |  все
|  кц
кон
```

21.13. Задачи поиска

Задачи поиска внешне очень похожи на задачи анализа. В них таблица тоже задана, т. е. является ***аргументом*** алгоритма, и нужно найти, *где* в этой таблице находится нужный элемент.

Существуют различные методы поиска, но мы рассмотрим самый простой из них — ***метод последовательного перебора***.

Пусть надо найти в таблице элемент, равный x. Найти элемент — значит найти его индекс, т. е. такое i, что $A[i] = x$. Поскольку искомого элемента может и не оказаться в таблице, будем считать, что в этом случае $i = 0$.

Простейший способ решения этой задачи — перебрать все элементы таблицы и запомнить, где расположен нужный.

А 88

```
алг поиск (арг цел n, вещ таб A[1:n], вещ x, рез цел i)
   надо | A[i] =x, если значение x встречается в таблице A,
        | i = 0 в противном случае
нач цел j
|  i := 0
|  нц для j от 1 до n
|  |  если A[j] = x
|  |  | то i := j
|  |  все
|  кц
кон
```

У этого алгоритма есть два недостатка. Во-первых, если значение x встречается в таблице несколько раз, мы получим в результате номер *последнего* из этих элементов. Во-вторых, после того как элемент найден, происходит уже ненужный просмотр таблицы до конца.

Для устранения этих недостатков достаточно организовать алгоритм так, чтобы остановиться сразу же после первой встречи с элементом, равным x. Но поскольку мы не знаем, где находится такой элемент и есть ли он вообще, мы не можем заранее определить количество повторений цикла. Следовательно, вместо цикла **для** необходимо использовать **пока**.

A 89

```
алг поиск (арг цел n, вещ таб A[1:n], вещ x, рез цел i)
    надо | A[i] = x, если значение x встречается в таблице A,
         | i = 0 в противном случае
нач
|   i := 1
|   нц пока i <= n и A[i] <> x
|   |   i := i + 1
|   кц
|   утв i > n или A[i] = x
|   если i > n
|   |   то i := 0
|   все
кон
```

Обратите внимание на составное условие в цикле **пока**. Оно обеспечивает проверку нахождения нужного элемента и одновременно не позволяет выйти за границу таблицы.

21.14. Задачи перестановки

В задачах перестановки таблица одновременно является ***аргументом*** и ***результатом***. Рассмотрим простой пример.

Пример. Необходимо переставить элементы таблицы следующим образом: второй поставить на первое место, третий — на второе и т. д. Последний элемент переходит на предпоследнее место, а первый становится последним. Такая перестановка называется ***циклическим сдвигом***. При решении этой задачи надо учесть, что, когда величина получает новое значение, старое теряется безвозвратно. При циклическом сдвиге все элементы таблицы получают новые значения, но в то же время их надо сохранить. Для этого необходима дополнительная ячейка памяти, в которой будет временно храниться значение первого элемента.

A 90

```
алг циклический сдвиг (арг цел n, арг рез вещ таб A[1:n])
нач цел i, вещ t
|   t := A[1]
|   нц для i от 1 до n - 1
|   |   A[i] := A[i + 1]
|   кц
|   A[n] := t
кон
```

21.15. Сортировка

Часто встречающаяся задача перестановки — это сортировка таблицы, т. е. расположение ее элементов по возрастанию (или по убыванию). Известны десятки различных методов сортировки, каждый из которых имеет свои достоинства и недостатки.

Рассмотрим один из простейших методов сортировки.

Чтобы упорядочить таблицу из n элементов, поступим так. Найдем в таблице минимальный элемент и поменяем его местами с первым элементом. Затем найдем минимальный среди элементов, начиная со второго, и поменяем его местами со вторым. И так далее: на очередном i-м шаге будем искать минимальный элемент среди всех, начиная с i-го, и менять его местами с i-м элементом. В результате получится таблица, в которой каждый элемент меньше всех тех, что идут после него, а это и есть ***упорядоченная***, или ***отсортированная, таблица***.

A 91

```
алг сортировка (арг цел n, арг рез вещ таб A[1:n])
нач цел i, m, вещ t
|   нц для i от 1 до n - 1
|   |   m := индекс минимума (n, A, i)
|   |   t := A[i]
|   |   A[i] := A[m]
|   |   A[m] := t
|   кц
кон
```

А 92

```
алг цел индекс минимума (арг цел n, вещ таб A[1:n], цел k)
    надо | знач = индекс минимального среди элементов
         | с индексами от k до n
нач цел i
|   знач := k
|   нц для i от k + 1 до n
|   |   если A[i] < A[знач]
|   |   то знач := i
|   |   все
|   кц
кон
```

21.16. Прямоугольные таблицы

Во всех таблицах, которые мы рассматривали до сих пор, элементы располагались по порядку, один за другим. Чтобы пронумеровать их, достаточно было одного индекса. Такие таблицы называются ***линейными***.

Прямоугольные таблицы отличаются от линейных только тем, что их элементы располагаются не в одну линию, а по двум направлениям в виде прямоугольника:

вещ таб b[1:3, 1:5]

1	7.15	2.71828	−4.56	−17	0
2	3.14	−9.81	10.11	23.25	−91
3	5.44	161	−4.5	256	123
	1	2	3	4	5

Чтобы указать элемент прямоугольной таблицы, надо использовать не один индекс, а два: номер строки и номер столбца, на пересечении которых находится элемент. Например, в приведенной таблице $b[3, 2] = 161$, а $b[2, 3] = 10.11$.

При описании таблицы надо указать диапазон изменения каждого индекса. Для приведенной выше таблицы b описание будет таким: **вещ таб** b[1:3, 1:5]. Это означает, что таблица b содержит 3 строки и 5 столбцов.

При решении задач с прямоугольными таблицами используются те же основные принципы, что и для линейных таблиц, но вместо одного цикла необходимо применять два вложенных цикла: по строкам и по столбцам прямоугольной таблицы.

Вот, например, как можно найти сумму всех элементов прямоугольной таблицы:

А 93

```
алг сумма (арг цел n, m, вещ таб A[1:n, 1:m], рез вещ S)
    надо | S = сумма элементов таблицы A
нач цел i, j
|   S := 0
|   нц для i от 1 до n
|   |   нц для j от 1 до m
|   |   |   S := S + A[i, j]
|   |   кц
|   кц
кон
```

ЗАДАЧИ И УПРАЖНЕНИЯ

1. Известно, что значением целочисленной табличной величины k является четверка чисел (3, 1, −2, 4). Чему равно:

   а) $k[1]$; б) $k[3]$; в) $k[5]$; г) $k[k[1]]$?

2. Известно, что значением целочисленной табличной величины k является четверка чисел (3, 1, −2, 4). Вычислите значение величины k после выполнения серии команд:

   а) k[1] := k[4]; k[1] := k[1] + k[2];

   б) i := 3; k[1] := k[i] * k[i + 1].

3. Задана линейная таблица $k[1:4]$. Запишите арифметическое выражение, значением которого является:
   а) удвоенное значение третьего элемента таблицы;
   б) сумма первого и второго элементов таблицы;
   в) произведение всех элементов таблицы.

4. Составьте алгоритмы со следующими заголовками:

   а) **алг** обмен (**арг цел** n, **арг рез вещ таб** a[1:n])
   надо | первый и последний элементы таблицы поменялись местами

б) **алг** перестановка (**арг цел** n, **арг рез вещ таб** a[1:n])
надо | минимальный элемент таблицы переставлен в ее
| начало, а первый — на место минимального

в) **алг вещ** минимум (**арг цел** n, **вещ таб** a[1:n])
надо | **знач** = значение минимального элемента в таблице a

г) **алг цел** индекс минимума (**арг цел** n, **вещ таб** a[1:n])
надо | **знач** = индекс минимального элемента в таблице a

д) **алг цел** число больших (**арг цел** n, **вещ таб** a[1:n], **вещ** x)
надо | **знач** = число элементов таблицы a, больших x

е) **алг** копия (**арг цел** n, **вещ таб** a[1:n], **рез вещ таб** b[1:n])
надо | таблица b содержит копию таблицы a

ж) **алг** копия (**арг цел** n, **вещ таб** a[1:n], **рез вещ таб** b[1:n])
надо | элементы таблицы a скопированы в таблицу b
| в обратном порядке: b[1] = a[n], b[2] = a[n – 1], ...

з) **алг** корректировка (**арг цел** n, **арг рез цел таб** a[1:n])
надо | все элементы таблицы a, меньшие 100, заменены
| на 100

5. Составьте алгоритмы со следующими заголовками:

а) **алг цел** произведение (**арг цел** n, **арг цел таб** a[1:n])
надо | **знач** = произведение элементов таблицы a

б) **алг вещ** среднее арифметическое (**арг цел** n, **вещ таб** a[1:n]
надо | **знач** = среднее арифметическое элементов a

в) **алг** график (**арг вещ** a, b, **цел** n, **вещ таб** y[1:n])
дано a < b **и** n >= 2 | в таблице y заданы n значений функции
| на отрезке [a, b] в равноотстоящих
| точках
надо | Чертежник построил приближенный график функции
| на отрезке [a, b] в виде ломаной с n вершинами

г) **алг** ломаная (**арг цел** n, **вещ таб** x[1:n], y[1:n])
дано n >= 2 | в таблицах x и y заданы координаты вершин
| ломаной линии с n вершинами
надо | Чертежник нарисовал эту ломаную линию

д) **алг** простые числа (**арг цел** n, **рез цел таб** p[1:n])
дано n > 1
надо | в таблице p — первые n простых чисел

е) **алг** упорядочение (**арг цел** n, **арг рез цел таб** a[1:n])
надо | элементы a упорядочены по невозрастанию,т. е.
| переставлены так, что a[1] >= a[2] >= ... >= a[n]

6. Составьте алгоритмы со следующими заголовками:

а) **алг** уровни радиации (**арг цел** m, n, **рез вещ таб** a[1:m, 1:n])
дано | Робот в левом верхнем углу прямоугольника
| размером m*n клеток, внутри стен нет
надо | в таблице a запомнены уровни радиации всех кле-
| ток прямоугольника, Робот в исходном положении

б) **алг вещ** произведение (**арг цел** m, n, **вещ таб** a[1:m, 1:n])
надо | **знач** = произведение элементов таблицы a

22 Символьные и литерные величины

22.1. Представление текстовой информации

Как мы знаем, обработка текстов — одно из важнейших применений современных компьютеров. Практически на любом компьютере есть текстовый редактор, часто это одна из самых используемых программ.

Чтобы обрабатывать текстовую информацию, надо уметь представлять ее в памяти компьютера. Мы уже знаем, что информация в памяти представляется в виде величин, при этом существуют различные типы величин, которые отличаются друг от друга допустимыми значениями, т. е. представляют различного рода информацию.

Для работы с текстовой информацией в алгоритмическом языке существует два типа величин: символьный (**сим**) и литерный (**лит**).

22.2. Символьные величины

Значением символьной (**сим**) величины является один символ: русская или латинская буква, цифра, знак препинания или специальный знак (например, «+», «–», «*», «/», «<», «=» и др.).

Существует также специальный символ « », который называется ***пробелом*** и используется для разделения слов в последовательности символов.

Обычно для хранения символа в памяти компьютера выделяется 1 байт, поэтому количество допустимых символов равно

256. Все эти символы пронумерованы от 0 до 255. Номер символа называется его ***кодом***.

В алгоритме ***символьные*** значения записываются в апострофах, например 'a', '+', '4'. Для символьных величин можно выполнять присваивание. Допускаются также сравнения символьных величин на равенство и неравенство.

Например, если в алгоритме описана величина **сим** s, в нем могут встретиться такие строки:

```
s := '4'
если s = '+' ...
нц пока s <> 'a' ...
```

22.3. Литерные величины

Обработка текста, как правило, не сводится к работе с отдельными символами, поэтому символьного типа во многих случаях оказывается недостаточно. Другой тип величин в алгоритмическом языке — литерный (**лит**) — позволяет представить сразу несколько символов.

Значением литерной величины является последовательность символов. Иногда эту последовательность называют строкой или цепочкой.

Литерные значения в алгоритме записывают в кавычках, например: "информатика", "крокодил", "литерная величина".

Литерная величина очень похожа на таблицу с базовым типом **сим**. Действительно, многие приемы работы с таблицами применимы и к литерным величинам. Но у литерных величин есть и специфические особенности, которых нет у обычных таблиц.

22.4. Длина литерной величины

Строка — значение литерной величины — состоит из символов. Количество символов называется ***длиной*** строки. Для нахождения длины строки используется встроенная функция ***длин***. Например, длин ("кот") = 3, длин ("два кота") = 8.

Существует особое значение литерной величины, которое называется ***пустая строка*** и не содержит ни одного символа. Пустая строка обозначается " " (между кавычками ничего нет), ее длина равна нулю.

Пустую строку надо отличать от строки " ", состоящей из одного пробела. Пробел — равноправный со всеми остальными символ, поэтому длин (" ") = 1, в то время как длин ("") = 0.

Литерная величина может изменять длину в процессе выполнения алгоритма. Например, если в алгоритме описана величина **лит** t, то после выполнения команды t := "что-нибудь" имеем длин (t) = 10. Если после этого выполнить команду t := "что-то другое", получим длин (t) = 13.

22.5. Работа с отдельными символами

С каждым из символов, составляющих литерную величину, можно работать отдельно. Символы литерной величины всегда пронумерованы, начиная с единицы: номер первого символа равен 1, номер последнего — длине литерной величины.

Обращение к отдельному символу выполняется по тем же правилам, что и к элементу таблицы: надо указать номер символа в квадратных скобках после имени величины. Номер символа в строке называется его ***индексом***.

Так, после выполнения команды t := "информатика" значение t можно представить в виде такой таблицы:

лит t

и	н	ф	о	р	м	а	т	и	к	а
1	2	3	4	5	6	7	8	9	10	11

Здесь *t* [1] = «и», *t* [2] = «н», *t* [3] = «ф», ..., *t* [11] = «а».

22.6. Вывод строки по вертикали

Задача. Вывести значение литерной величины по вертикали — по одной букве в каждой строчке экрана.

Чтобы сделать это, переберем все символы заданной величины, начиная с первого, и будем выводить каждый из них с новой строки. Для организации перебора используем цикл **для**.

A 94

```
алг вертикальная печать (арг лит s)
    надо | строка s напечатана по вертикали
нач цел i
|   нц для i от 1 до длин (s)
|   |   вывод нс, s[i]
|   кц
кон
```

22.7. Метод посимвольной обработки

Многие задачи сводятся к посимвольной обработке заданной литерной величины. Обычно в таких задачах нужно по одному перебрать все символы, входящие в строку, и выполнить с каждым из них какие-то действия. По этой схеме построен, например, алгоритм "вертикальная печать".

Вот как выглядит общая схема алгоритма посимвольной обработки:

```
нц для i от 1 до длин (s)
|  обработка (s[i])
кц
```

Метод посимвольной обработки — это не что иное, как метод однопроходных алгоритмов, примененный к литерным величинам.

22.8. Сколько раз в строке встречается символ *x*

A 95

```
алг цел количество символов (арг сим x, арг лит a)
   надо | знач = количество символов x в строке a
нач цел i
|  знач := 0
|  нц для i от 1 до длин (a)
|  |  если a[i] = x
|  |  |  то знач := знач + 1
|  |  все
|  кц
кон
```

Чтобы с помощью этого алгоритма узнать общее количество k восклицательных и вопросительных знаков в строке b, достаточно написать команду присваивания:

```
k := количество символов("!", b) + количество символов ("?", b)
```

22.9. Доля пробелов в строке

Для определения доли пробелов в строке надо число пробелов поделить на общее количество символов (длину строки). Воспользуемся для подсчета числа пробелов алгоритмом A95:

A 96

```
алг вещ доля пробелов (арг лит a)
   дано длин (a) > 0
   надо | знач = доля пробелов в строке a
нач
|  знач := количество символов (" ", a)/длин (a)
кон
```

22.10. Замена одного символа на другой

Обратившись к отдельному символу литерной величины, можно не только проанализировать, но и изменить его, точно так же как можно изменить значение одного элемента простой таблицы.

Например, если выполнить команды

```
t := "слон"
t[2] :="т"
```

то в результате литерная величина t получит значение "стон".

С помощью этого приема легко, например, выполнить замену одного символа на другой во всей строке.

A 97

```
алг замена (арг сим x, y, арг раз лит a)
   надо |  в строке a все символы x заменены на y
нач цел i
|  нц для i от 1 до длин (a)
|  |  если a[i] = x
|  |  |  то a[i] := y
|  |  все
|  кц
кон
```

В алгоритме A97 величина a является одновременно аргументом и результатом (отсюда и запись **арг рез лит** a). Пусть, например,

```
b = "Понял. Немедленно выезжаю."
```

Тогда после вызова

```
замена (".", "!", b)
```

получим:

```
b = "Понял! Немедленно выезжаю!"
```

А если теперь вызвать замена ("е", "и", b), получим

```
b = "Понял! Нимидлинно выизжаю!"
```

22.11. Операция соединения

Для литерных величин существует специальная операция — соединение. Она обозначается знаком «+». При выполнении соединения двух строк получается новая строка, образованная приписыванием второй строки в конец первой. Например,

"бар" + "сук" = "барсук".

Обратите внимание, что, хотя для соединения используется тот же знак, что и для арифметического сложения, это совершенно разные операции. При соединении «сумма» зависит от порядка «слагаемых». Например, если y и z — литерные величины, y = «вес», z = «на», то $y + z$ = «весна», но $z + y$ = «навес».

Еще одно важное отличие соединения от сложения — отсутствие обратной операции — никакого аналога вычитания для литерных величин не существует.

22.12. Метод посимвольного формирования

Во многих задачах требуется на основе заданной литерной величины построить другую.

Чаще всего подобные задачи легко решаются методом посимвольного формирования. Суть его в следующем. Берется пустая строка, к которой постепенно с помощью операции соединения добавляются символы, полученные в результате анализа исходной строки.

Метод посимвольного формирования особенно удобен, когда в результате преобразования должна измениться длина исходной строки.

Это хорошо иллюстрирует следующий алгоритм:

A 98

```
алг вычеркивания (арг сим x, арг рез лит a)
   надо | в строке a все символы x вычеркнуты
нач цел i, лит b
|  b := ""
|  нц для i от 1 до длин (a)
|  |  если a[i] <> x
|  |  то b := b + a[i]
|  |  все
|  кц
|  a := b
кон
```


В алгоритме A98 исходная строка (до вычеркивания) не сохраняется. Если требуется при посимвольном формировании сохранить исходную строку, то ее надо сделать аргументом алгоритма, как это показано в алгоритмах A99, A100.

A 99

```
алг каждый второй (арг лит a, рез лит b)
   дано mod (длин (a), 2) = 0 | в a четное число символов
   надо | строка b состоит из «четных» символов (2-й, 4-й, ...)
        | строки a
нач цел i
|  b := "" | пустая строка
|  нц для i от 1 до длин (a)/2
|  |  b := b + a[i*2]
|  кц
кон
```

A 100

```
алг вычеркивание пробелов (арг лит a, рез лит b)
   надо | строка b получена из a вычеркиванием всех пробелов
нач цел i
|  b := "" | пустая строка
|  нц для i от 1 до длин (a)
|  |  если a[i] <> " " | пробел
|  |  то b := b + a[i]
|  |  все
|  кц
кон
```

В приведенных алгоритмах новая строка формировалась последовательным добавлением символов в конец строки. Иногда удобно формировать строку в обратном порядке — от конца к началу. Вот типичный пример подобной задачи:

A 101

```
алг переворот (арг лит a, рез лит b)
   надо | b содержит строку a, прочитанную с конца
нач цел i
|  b := " "
|  нц для i от 1 до длин (a)
|  |  b := a[i] + b
|  кц
кон
```

22.13. Вырезки

До сих пор мы работали со строками либо целиком, либо поэлементно. Операция ***вырезки*** позволяет «вырезать» из строки группу соседних символов.

Вырезка из строки *t* обозначается t[i:j] (читается «*t* от *i* до *j*»), где *i* — индекс первого, а *j* — индекс последнего элементов вырезки. Например, если значением литерной величины *t* является строка "информатика", то

t[1:4] = "инфо" t[1:4] + "МИР" = "инфоМИР"
t[8:10] = "тик" t[10:10] = "к"
t[10:10] + t[1:2] + t[4] = "кино"

22.14. Команда присваивания вырезке

Вырезку можно использовать в левой части команды присваивания. Так, если t = "План недовыполнен", то при выполнении команды присваивания

t[6:9] := "пере"

элементы строки *t* с шестого по девятый будут заменены на "пере" и *t* станет равным "План перевыполнен».

Пример алгоритма, использующего вырезки:

А 102

```
алг замена (арг рез лит t)
   надо | в строке t "1990" всюду заменено на "2000"
нач цел i
|  нц для i от 1 до длин (t) – 3
|  |  если t[i:i + 3] = "1990"
|  |  |  то t[i:i + 3] := "2000"
|  |  все
|  кц
кон
```

ЗАДАЧИ И УПРАЖНЕНИЯ

1. Строка t равна "апельсин". Чему равны строки:
   а) t[2:3] + t[8] + t[5]; б) t[6] + t[2] + t[1] + t[8] + t[7] + t[3:5]?
2. Строка t равна "вертикаль". По образцу упражнения 1 составьте из *t* слова: а) «ветка»; б) «кирка»; в) «кильватер».

3. Составьте алгоритмы со следующими заголовками:

   а) **алг цел** количество предложений (**арг лит** x)
   дано | строка x состоит из нескольких предложений,
   | каждое из которых кончается точкой, восклицательным или вопросительным знаком
   надо | **знач** = количество предложений в строке x

   б) **алг** обмен (**арг сим** x, y, **арг рез лит** t)
   надо | всюду в "t" x заменен на y, а y заменен на x

   в) **алг лит** перевертыш (**арг лит** t)
   надо | **знач** = "перевертыш", если строка t совпадает
   | с собой после переворачивания,
   | **знач** = "не перевертыш" в противном случае

   г) **алг** замена (**арг лит** x, y, **арг рез лит** t)
   дано длин (x) = длин (y)
   надо | всюду в строке t x заменено на y

4. Строка состоит из нескольких слов, разделенных пробелами. Составьте алгоритмы для решения следующих задач:
   а) подсчитать количество слов в строке;
   б) удалить из строки лишние пробелы, т. е. оставить только один пробел там, где подряд идет несколько пробелов;
   в) удалить первую букву каждого слова;
   г) удалить последнюю букву каждого слова;
   д) вывести на экран все слова из строки, по одному в каждой строчке;
   е) вывести на экран самое длинное слово в строке.

5. Составьте алгоритмы, которые из строки a получают зашифрованную строку b, по следующим правилам:
   а) каждая буква от «а» до «ю» заменяется на следующую по алфавиту, каждая буква «я» заменяется на букву «а»;
   б) первая буква алфавита заменяется на одиннадцатую, вторая на двенадцатую, ..., последняя — на десятую;
   в) после каждой согласной вставляется буква «а»;
   г) после каждой согласной вставляется слог «ла».

6. Составьте алгоритмы, которые из зашифрованной строки b получают расшифрованную строку a (см. упражнение 5).

7. Придумайте способ шифровки, при котором каждая пара букв заменяется либо одной, либо тремя буквами. Составьте алгоритмы шифровки и дешифровки для этого способа.

ГЛАВА 4

Устройство компьютера

23 Двоичная система счисления

23.1. Как мы считаем и записываем числа

Мы привыкли считать десятками. Степени десяти — числа десять (10^1), сто (10^2), тысяча (10^3), миллион (10^6) мы воспринимаем как круглые, особенно удобные для счета.

Такой способ счета и такое восприятие круглых чисел сложились исторически и связаны в первую очередь с анатомическими особенностями человека. У человека на руках 10 пальцев, а поскольку пальцы были первым естественным счетным приспособлением, то число 10 утвердилось в качестве основы для счета. Возможно, если бы мы имели по 6, а не по 5 пальцев на каждой руке, то в основе счета оказалось бы число 12.

Способ счета еще не определяет правил записи чисел. Например, широко известный римский способ записи чисел тоже основан на счете десятками, но римская запись сильно отличается от обычной. И дело даже не в том, что в римской записи для чисел используются не специальные символы — цифры, а буквы латинского алфавита. Главное свойство римской записи заключается в том, что в ней каждый символ всегда обозначает одно и то же

число (табл. 13). Например, XX в римской записи обозначает 20, CCC — 300 и т. д. Такая система называется ***непозиционной***.

ТАБЛИЦА 13. **Римские цифры**

I	V	X	L	C	D	M
1	5	10	50	100	500	1000

В привычном нам способе записи одна и та же цифра соответствует разным значениям в зависимости от того, на каком месте (в какой позиции) числа она стоит. Например, в числе 777 первая семерка обозначает 700, вторая — 70, а третья — 7. Подобная система записи называется ***позиционной***.

23.2. Правила записи десятичных чисел

Привычная нам система записи чисел называется ***десятичная позиционная система***. Слово *позиционная* показывает, что значение каждой цифры зависит от ее положения в числе, слово *десятичная* — что в основе системы лежит число десять.

Рассмотрим правила записи чисел в десятичной системе более подробно. Записывая число, мы фактически представляем его в виде суммы степеней числа 10 — ***основания системы счисления***. Например, запись 1998 означает сумму одной тысячи, девяти сотен, девяти десятков и восьми единиц, что можно записать так:

$$1998 = 1*10^3 + 9*10^2 + 9*10^1 + 8*10^0.$$

Последняя цифра любого числа — это количество единиц, вторая с конца — количество десятков и т. д. Каждая цифра имеет тем больший вес, чем дальше от конца она находится.

При делении числа на 10 в остатке получается его последняя цифра, а в частном — все число без последней цифры. Повторяя такое деление многократно, мы можем получить все цифры числа (см. п. 17.6, алгоритм А65).

23.3. Алгоритмы преобразования десятичных чисел

При составлении алгоритмов мы используем для целых чисел тип **цел**, не думая о том, как эти числа представлены в памяти

компьютера. Выполняя команды **ввод** и **вывод**, компьютер преобразует числа в последовательность символов, образующих привычную нам десятичную запись. Понимая, как устроена десятичная система, мы можем составить алгоритмы, выполняющие эти преобразования.

A 103

```
алг записать число (арг цел n, рез лит s)
    дано n >= 0
    надо | s содержит десятичную запись числа n
нач цел nn | копия числа n необходима, потому что в алгоритме
           | нельзя изменять значение аргумента
    nn := n
    если n = 0
    | то s := "0"
    | иначе s := "" | пустая строка
    все
    нц пока nn > 0
    | s := цифра (mod(nn, 10)) + s
    | nn := div(nn, 10)
    кц
кон
```

A 104

```
алг сим цифра (арг цел k)
    дано 0 <= k <= 9
    надо | знач содержит символ цифры, соответствующей
         | значению k
нач
    выбор
    | при k = 0 : знач := '0'
    | при k = 1 : знач := '1'
    | при k = 2 : знач := '2'
    | при k = 3 : знач := '3'
    | при k = 4 : знач := '4'
    | при k = 5 : знач := '5'
    | при k = 6 : знач := '6'
    | при k = 7 : знач := '7'
    | при k = 8 : знач := '8'
    | при k = 9 : знач := '9'
    все
кон
```


A 105

```
алг прочитать число (арг лит s, рез цел n)
    дано длин (s) > 0 | все символы в s — цифры от 0 до 9
    надо | n содержит число, записанное в s
нач цел i
    n := 0
    нц для i от 1 до длин (s)
    | n := 10*n + значение цифры (s[i])
    кц
кон
```

Алгоритм-функция "значение цифры" очень похож на алгоритм "цифра" (А104), но выполняет обратное преобразование — из символа в число. Вы можете составить его самостоятельно.

23.4. Позиционные системы с другими основаниями

Мы можем представить любое число в виде суммы степеней не 10, а какого-то другого основания, сохранив при этом правила позиционной системы. При этом мы будем получать запись числа в недесятичных позиционных системах счисления.

Например, взяв в качестве основания числа 3, 5, 12, мы получим троичную, пятеричную, двенадцатеричную системы.

Алгоритмы преобразования А103 и А105 будут правильно работать для любой системы счисления. Вспомогательные алгоритмы "цифра" и "значение цифры" придется изменить. В системе счисления с основанием M используется M различных цифр, поэтому при $M < 10$ из этих алгоритмов надо будет убрать преобразование цифр, лишних для данной системы, а при $M > 10$ — добавить преобразование дополнительных цифр.

Если нужно указать, в какой системе записано число, то после числа в виде индекса указывается основание системы счисления. Само основание при этом записывается в десятичной системе. Например, 23_5, 10011_2, 212_3, 997_{10}.

23.5. Двоичная система счисления

Минимально возможное основание для позиционной системы счисления равно 2. Такая система называется ***двоичной***. В двоичной системе всего две цифры, поэтому алгоритмы преобразования для нее можно сделать очень простыми.

A 106

```
алг записать двоичное число (арг цел n, рез лит s)
    дано n >= 0
    надо | s содержит двоичную запись числа n
нач цел nn | копия числа n необходима, потому что в алгоритме
           | нельзя изменять значение аргумента
    nn := n
    если n = 0
    | то s := "0"
    | иначе s:= ""   | пустая строка
    все
    нц пока nn > 0
    | если mod(nn, 2) = 1
    | | то s := '1' + s
    | | иначе s := '0' + s
    | все
    | nn := div(nn, 2)
    кц
кон
```

A 107

```
алг прочитать двоичное число (арг лит s, рез цел n)
    дано длин (s) > 0  | все символы в s — цифры 0 или 1
    надо  | n содержит двоичное число, записанное в s
нач цел i
    n := 0
    нц для i от 1 до длин (s)
    | n := 2*n
    | если s[i] = '1'
    | | то n := n + 1
    | все
    кц
кон
```

23.6. Преобразование двоичных чисел вручную

Для ручного преобразования двоичных и десятичных чисел можно, конечно, воспользоваться алгоритмами А106 и А107. Но эти алгоритмы, эффективные при выполнении на компьютере, не совсем удобны для человека. При ручном преобразовании удобнее применять другой способ, основанный на использовании таблицы степеней двойки (табл. 14) и представлении числа в виде суммы этих степеней.

ТАБЛИЦА 14. **Степени двойки**

n	0	1	2	3	4	5	6	7	8
2^n	1	2	4	8	16	32	64	128	256
n	9	10	11	12	13	14	15	16	
2^n	512	1024	2048	4096	8192	16 384	32 768	65 536	

Рассмотрим схему преобразования на примере. Переведем число 1998 в двоичную систему. Для этого найдем в таблице наибольшую степень двойки, не превосходящую заданного числа, и вычтем ее из исходного числа, а с тем, что останется, будем проделывать ту же операцию, пока не получим нуль.

$$\begin{aligned} 1998 &= 1024 + 976 \\ 976 &= 512 + 464 \\ 464 &= 256 + 208 \\ 208 &= 128 + 80 \\ 80 &= 64 + 16 \\ 16 &= 16 + 0 \end{aligned}$$

$$1998 = 1024 + 512 + 256 + 128 + 64 + 16 = \\ = 2^{10} + 2^9 + 2^8 + 2^7 + 2^6 + 2^4$$

Следовательно, двоичная запись числа 1998 — 11111010000. Коротко это можно записать так: $1998_{10} = 11111010000_2$

Обратите внимание на нули, которые появляются в двоичной записи на месте тех степеней двойки, которые не вошли в сумму.

Рассмотрим схему обратного преобразования. Переведем в десятичную запись 1011011101. Для этого запишем двоичное число как сумму степеней двойки, возьмем из таблицы десятичные значения этих степеней и сложим их.

$$1011011101_2 = 2^9 + 2^7 + 2^6 + 2^4 + 2^3 + 2^2 + 2^0 = \\ = 512 + 128 + 64 + 16 + 8 + 4 + 1 = 733_{10}$$

23.7. Двоичные числа в вычислительной технике

Двоичная система содержит всего две цифры. Эти цифры легко представить в виде двух различных сигналов (например, высокое и низкое напряжение). Память компьютера состоит из эле-

ментов, которые могут находиться в одном из двух устойчивых состояний, поэтому двоичная система оказывается самой удобной для компьютерного представления чисел.

Мы рассмотрели правила двоичной записи только для натуральных чисел. Именно по этим правилам кодируются натуральные числа в памяти современных компьютеров. Существуют также специальные приемы, позволяющие представить в двоичном коде отрицательные, а также дробные числа.

ЗАДАЧИ И УПРАЖНЕНИЯ

1. Составьте вспомогательный алгоритм "значение цифры" для алгоритма А105.
2. Составьте алгоритмы преобразования из числа в строку и обратно для троичной и пятеричной систем счисления.
3. Переведите в двоичную систему счисления:
   а) число 2053;
   б) год своего рождения;
   в) день и месяц вашего рождения, записанные подряд как единое число (например, 905 — девятое мая).
4. Запишите свою фамилию в виде «двоичного кода»: вместо согласных букв поставьте 1, а вместо гласных — 0. Переведите полученное двоичное число в десятичную систему.
5. Даны две строки, содержащие записи двоичных чисел. Составьте алгоритм, результатом которого будет строка, содержащая сумму этих чисел.
6. Решите задачу 5, не используя преобразование строки в числовое представление.

24 Физические основы вычислительной техники

24.1. Кодирование информации электрическими сигналами

Как вы уже знаете, информация хранится и обрабатывается в компьютере в двоичном виде — в виде последовательностей нулей и единиц. В современных компьютерах широко распространено представление «1» высоким напряжением (например, +5 В), а «0» — низким (около 0 В).

24.2. Электронный ключ

Основной элемент современных компьютеров — транзистор. В радиоаппаратуре транзисторы чаще всего используются для усиления плавно меняющихся сигналов. В вычислительной технике транзисторы работают в так называемом ***ключевом*** режиме. В этом режиме транзистор можно представлять как обычный выключатель, который в одном положении проводит ток (замкнут), а в другом — нет (разомкнут). Однако, в отличие от бытового выключателя, включение и выключение транзистора производится также с помощью электричества.

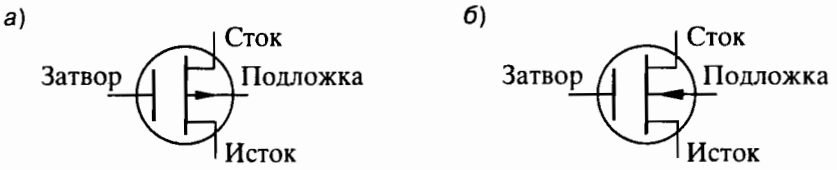


Рис. 89. Изображение МОП-транзисторов
а) *p*-канального; *б*) *n*-канального

Существует много типов транзисторов. Мы рассмотрим только ***МОП-транзисторы*** (рис. 89; сокращение МОП — металл-оксид-полупроводник — относится к устройству транзистора). Каждый МОП-транзистор имеет четыре контакта: сток, исток, затвор и подложку. *N*-канальный транзистор проводит ток между истоком и стоком (замкнут), только если на затвор подано положительное (относительно подложки) напряжение. Наоборот, *p*-канальный транзистор проводит ток, только если напряжение на затворе меньше напряжения на подложке.

24.3. Вентиль «не»

В качестве элементарных строительных «кубиков» обычно используются не сами транзисторы, а более крупные элементы, которые называются ***вентили***. Существуют разные способы объединения транзисторов в вентили.

Рассмотрим один из способов построения вентиля «не» (рис. 90). Если на его вход подать напряжение +5 В («1»), то нижний (*n*-канальный) транзистор замкнется, а верхний будет

разомкнут (рис. 91, *а*), на выходе вентиля установится напряжение 0 В («0»).

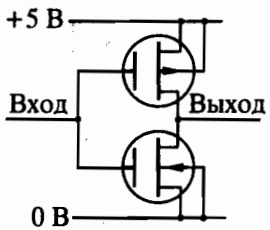
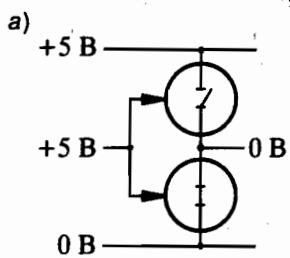


Рис. 90



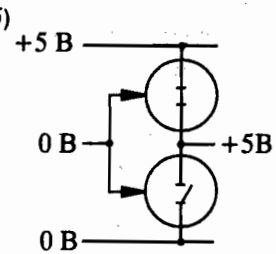


Рис. 91

Если же на вход вентиля подать 0 В, то замкнется верхний (*p*-канальный) транзистор, поскольку напряжение на его затворе (0 В) меньше напряжения на подложке (+5 В). На нижнем же транзисторе напряжения на затворе (0 В) и подложке (0 В) будут совпадать и транзистор разомкнется (рис. 91, *б*). В результате на выходе вентиля будет +5 В («1») (табл. 15).

ТАБЛИЦА 15. **Работа вентиля «не»**

Вход	Нижний транзистор			Верхний транзистор			Выход
	Затвор	Подложка	Состояние	Затвор	Подложка	Состояние	
+5 В	+5 В	0 В	Замкнут	+5 В	+5 В	Разомкнут	0 В
0 В	0 В	0 В	Разомкнут	0 В	+5 В	Замкнут	+5 В

Таким образом, рассмотренный вентиль можно считать простейшим электронным устройством обработки информации — он инвертирует («переворачивает») бит: «1» (+5 В) на входе превращает в «0» (0 В) на выходе, а «0» — в «1». Если сопоставить «1» значение **да**, а «0» — **нет**, то работу этого вентиля можно описать формулой

выход = **не** вход

Поэтому рассмотренный вентиль называется инвертором или вентилем «не».

24.4. Вентиль «или-не»

В компьютерах используются также другие типы вентилей, соответствующие другим элементарным операциям по обработке информации. На рисунке 92 изображен вентиль для формулы

выход = **не** (вход1 **или** вход2)

Состояния транзисторов (замкнут/разомкнут) и значение на выходе вентиля «или-не» для всех сочетаний значений входов изображены на рисунке 93.

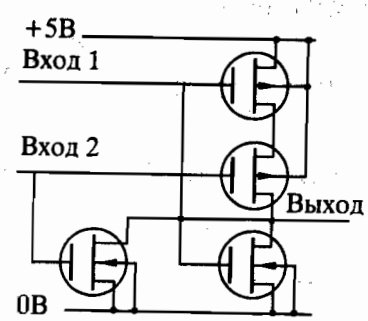


Рис. 92

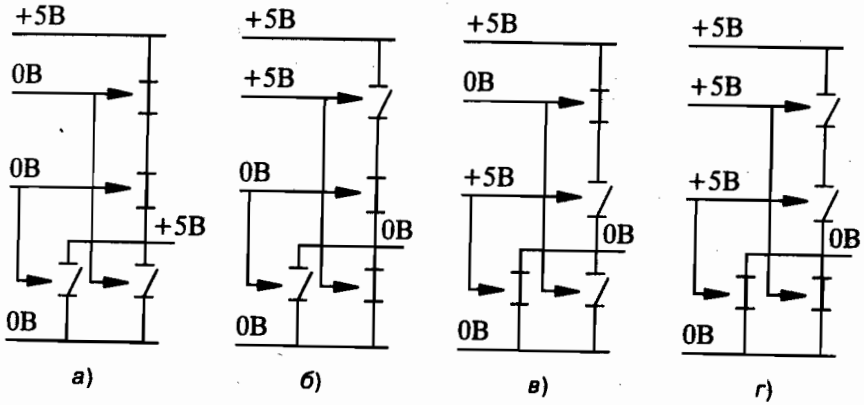


Рис. 93

24.5. Обозначения вентилей

В электронике наиболее широко распространены вентили пяти типов:

а) «и» : выход = вход1 **и** вход2;
б) «или» : выход = вход1 **или** вход2;
в) «не» : выход = **не** вход;
г) «и-не» : выход = **не** (вход1 **и** вход2);
д) «или-не» : выход = **не** (вход1 **или** вход2).

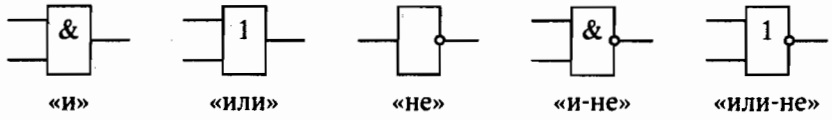


Рис. 94

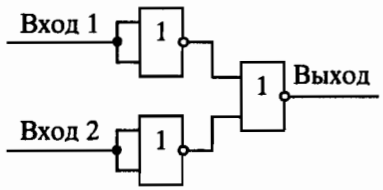


Рис. 95

Чтобы не загромождать схемы лишними деталями, для изображения вентилей применяются специальные обозначения (рис. 94).

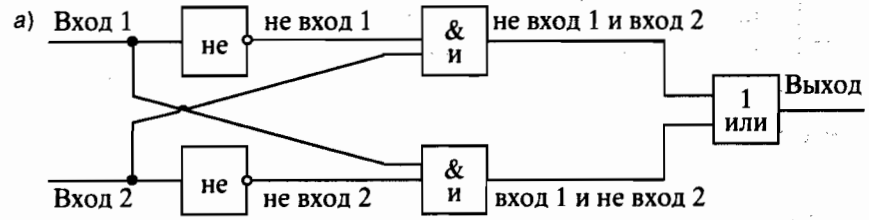
Приведенный набор вентилей является избыточным — одни вентили в нем выражаются через другие. Используя только вентили типа «или-не», например, можно собрать любую схему, в частности вентиль «и» (рис. 95).

24.6. Вентильные схемы для логических выражений

Легко заметить, что преобразование сигналов в вентилях описывается обычными логическими выражениями. Комбинируя вентили, можно составлять схемы для вычисления логических выражений любой сложности. Если на вход такой схемы подать сигналы «0» или «1», то сигнал на выходе будет соответствовать значению выражения при указанных входных данных.

Например, на рисунке 96 показана схема, которая описывается формулой выход = (вход1 **или** вход2) **и не** (вход1 **и** вход3).

Рис. 96



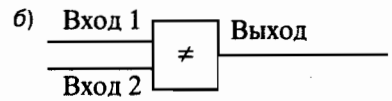


Рис. 97

С помощью вентильных схем можно описывать и операции сравнения. Построим, например, схему для формулы выход = = вход1 ≠ вход2. Для этого составим сначала таблицу истинности, содержащую значение выхода для всех возможных комбинаций входа (табл. 16). Эту таблицу можно задать формулой

выход = ((**не** вход1) **и** вход2) **или** (вход1 **и не** вход2)).

Схема для этого выражения изображена на рисунке 97, *а*. В дальнейшем всю эту схему мы будем изображать так, как показано на рис. 97, *б*.

ТАБЛИЦА 16. **Таблица истинности для функции выход = вход1 ≠ вход2**

вход1	вход2	выход = вход1 ≠ вход2
0	0	0
0	1	1
1	0	1
1	1	0

Таким образом, с помощью вентильных схем можно строить схемы различных логических выражений, в том числе включающих операции сравнения. Поэтому вентильные схемы часто называют ***логическими***.

24.7. Вычисление арифметических выражений с помощью логических схем

С помощью логических схем можно выполнять арифметические операции с двоичными числами. Рассмотрим, как это делается, на примере операции сложения однозначных (в вычислительной технике используется термин ***одноразрядных***) чисел.

Двоичная таблица сложения описывается всего четырьмя формулами:

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 10$$

Видно, что сумма двух одноразрядных чисел может быть одно- и двухразрядной. В таблице 17 показаны значения младшего S и старшего P разряда суммы в зависимости от значений исходных чисел $x1$ и $x2$.

ТАБЛИЦА 17. **Таблица истинности для одноразрядного сумматора**

$x1$	$x2$	S	P
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Получившуюся таблицу вполне можно рассматривать как таблицу истинности и записать значения S и P в виде логических выражений, зависящих от $x1$ и $x2$:

$$S = x1 \neq x2$$
$$P = x1 \text{ и } x2$$

Вентильная схема для этих выражений показана на рисунке 98, *а*. Эта схема называется ***полусумматор***.

Из нескольких полусумматоров можно собрать полный сумматор, выполняющий сложение двоичных чисел. Многоразрядное сложение напоминает школьный способ сложения «столбиком»: числа в каждом разряде складываются независимо, а если результат не помещается в одном разряде, то к следующему по

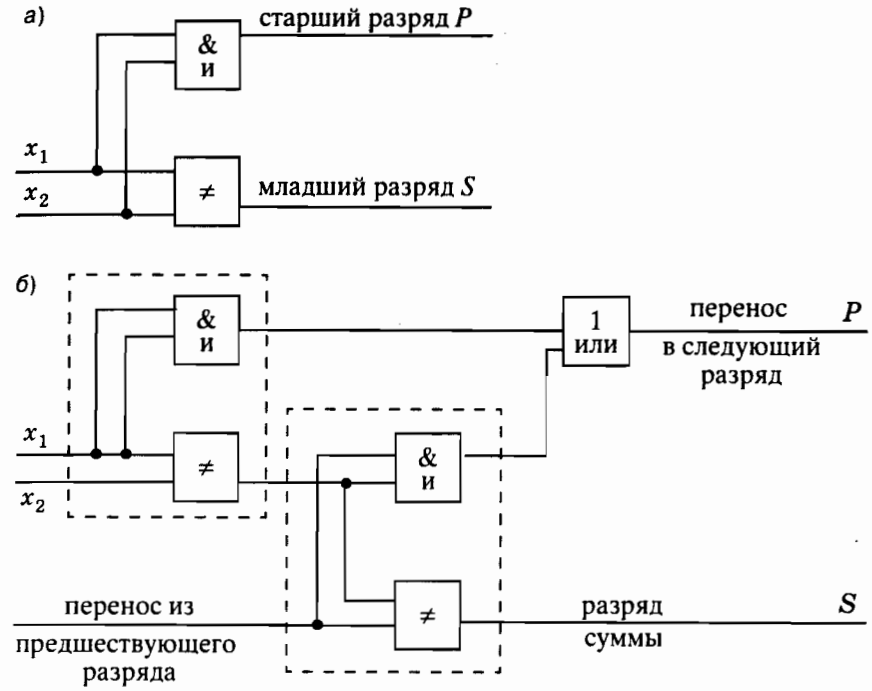


Рис. 98

старшинству разряду прибавляется единица. Этой единице соответствует сигнал P в схеме полусумматора. Элемент такого многоразрядного сумматора (для одного разряда) приведен на рисунке 98, *б*. Он составлен из *двух* полусумматоров (на рисунке 98, *б* обведены пунктиром), что и объясняет смысл слова «полусумматор».

Аналогичным образом можно свести к логическим выражениям и другие арифметические операции. На представлении арифметических операций в виде логических выражений основаны вычислительные возможности всех современных компьютеров.

24.8. Процессор

Обработку информации в компьютере производит процессор, который выполняет команды программы, хранящиеся в памяти. Эти команды выполняются логическими схемами, состоящими из десятков и сотен тысяч вентилей.

Обычно команда обрабатывает информацию не по одному биту, как в нашем простейшем примере, а одновременно группами по 8, 16, 32 или 64 бита.

Число одновременно обрабатываемых бит называется ***разрядностью процессора*** и является одной из важнейших характеристик компьютера.

Вместе с ***быстродействием*** (числом выполняемых команд в секунду) разрядность характеризует объем информации, перерабатываемой процессором компьютера в единицу времени.

24.9. Элемент памяти (триггер)

Приведенные примеры демонстрируют возможность обработки информации на компьютере. Покажем теперь, как можно информацию запоминать. Поскольку любая информация в компьютере представляется в двоичном виде, рассмотрим запоминание и хранение элементарной порции информации — одного бита. Электронная схема, запоминающая один бит информации, называется ***триггером***.

Существуют различные схемы триггеров, мы опишем один из самых простых, так называемый *RS*-триггер (рис. 99).

Если на входы этого триггера подать $S = 1, R = 0$, то (независимо от состояния Q) на выходе P верхнего вентиля появится 0. После этого на входах нижнего вентиля окажется $R = 0, P = 0$ и выход Q станет равным 1.

Если теперь перестать подавать сигналы на триггер ($S = 0$, $R = 0$), то, поскольку входами верхнего вентиля являются $S = 0$ и $Q = 1$, его выход P останется 0. Аналогично, поскольку входами нижнего вентиля являются $R = 0$ и $P = 0$, его выход Q по-прежнему будет «1». Таким образом, установленные значения выходов P и Q не изменятся при переходе к $S = 0, R = 0$.

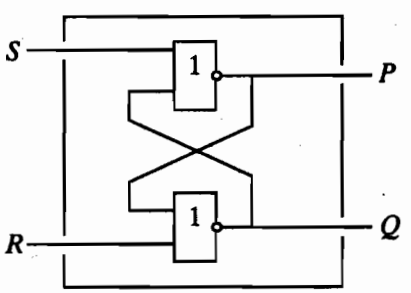


Рис. 99. *RS*-триггер

Точно так же при подаче $S = 0, R = 1$ на выходах появятся $Q = 0, P = 1$, и эти значения выходов сохранятся при снятии «1» со входа R ($R = 0, S = 0$).

Таким образом, при $S = 0, R = 0$ триггер может находиться в двух разных состояниях: $Q = 1$ и $Q = 0$. Выход Q и является значением запомненного бита. При подаче сигнала $S = 1$ на выходе устанавливается значение $Q = 1$, а при подаче $R = 1$ устанавливается $Q = 0$ (табл. 18).

ТАБЛИЦА 18. **Работа *RS*-триггера**

Вход S	Вход R	Действие триггера	Выход Q
1	0	Запоминание 1	1
0	1	Запоминание 0	0
0	0	Хранение бита	Запомненный бит

Одновременная подача сигналов $R = 1$ и $S = 1$ считается ошибкой, в правильно спроектированных схемах такого происходить не должно. В других, более сложных триггерах возможность подобной ошибки исключена.

Поскольку один триггер запоминает один бит, то для запоминания байта (8 бит) нужно 8 триггеров, для запоминания килобайта 1024*8 = 8192 триггера и т. д. Современные микросхемы памяти объемом менее 1 см^3 способны запоминать миллионы бит информации.

24.10. Память

В памяти компьютера запоминающие элементы обычно объединяют в группы из нескольких (8, 16, 32, 64) бит. Чаще всего используют группы из 8 бит (байты), которые занумерованы последовательными числами, начиная от 0. Например, в школьном компьютере УКНЦ, который выпускался в начале 90-х гг., байты пронумерованы от 0 до 65 535. Эти числа называются ***адресами*** байтов. Для кодирования адреса в УКНЦ используются после-

довательности из 16 бит. Число бит, с помощью которых кодируется адрес, называется ***разрядностью адреса*** компьютера. Это число характеризует максимальный объем памяти (информации), одновременно доступный процессору.

24.11. Взаимодействие процессора и памяти

Взаимодействие процессора и памяти сводится в основном к двум операциям: ***запись*** информации в память и ***чтение*** информации из памяти. При записи процессор по специальным проводникам (они называются ***шиной адреса***) передает биты, кодирующие адрес; по другим проводникам передает сигнал «запись» и по еще одной группе проводников (она называется ***шиной данных***) передает записываемую информацию. Конечно, слово «передает» означает просто, что схемы процессора подают на проводники напряжения, соответствующие «0» и «1». При чтении процессор устанавливает сигналы на шине адреса и считывает информацию с шины данных.

Число одновременно передаваемых по шине адреса и шине данных разрядов (бит) называется ***разрядностью*** соответствующей шины и является важной характеристикой компьютера. Разрядность шины адреса определяет максимальное общее количество доступной памяти; разрядность шины данных — максимальную порцию информации, которую можно получить из памяти за один раз.

Так, в первых компьютерах класса IBM PC процессор Intel 8088 был 16-разрядным, адрес — 16-разрядным, шина адреса — 20-разрядной, а шина данных — 8-разрядной. Поэтому для получения 16 разрядов данных, которые процессор мог обработать за одну команду, он должен был обратиться к памяти дважды.

24.12. Поколения ЭВМ

История развития компьютеров насчитывает несколько поколений. Смена поколений связана прежде всего с изменением физических принципов работы и технологией производства элементов, входящих в компьютер.

Слово «компьютер» стало общепринятым в русском языке сравнительно недавно, а до этого в течение долгого времени употреблялся термин ЭВМ — сокращение от слов «электронная вычислительная машина». Поэтому, говоря об истории, мы будем использовать это слово вместо современного «компьютер».

В ЭВМ *первого поколения* использовались электронные лампы (радиолампы) — те самые, которые еще можно найти в старых телевизорах. Помимо ламп, ЭВМ включали в себя другие элементы (резисторы, конденсаторы и т. п.). В настоящее время ЭВМ первого поколения не применяются.

Элементную базу ЭВМ *второго поколения* составляли транзисторы, изготовлявшиеся в виде отдельных деталей (каждый в своем корпусе). Они занимали меньше места, чем радиолампы, потребляли меньше энергии и были надежнее. Это позволило сделать ЭВМ более компактными, дешевыми и экономичными. Ко второму поколению ЭВМ относится одна из самых известных советских ЭВМ — БЭСМ-6, которая использовалась вплоть до середины 90-х гг.

Третье поколение ЭВМ обязано своим возникновением революционной идее изготовить фрагмент электронной схемы, содержащий много транзисторов, резисторов, конденсаторов, проводников и т. д., в виде участков различных веществ на поверхности одной полупроводниковой (обычно кремниевой) пластинки размером с клеточку школьной тетради. Такие электронные устройства получили название ***интегральных схем*** (от англ. *integrate* — собирать вместе, объединять), или ***микросхем***.

Одними из первых ЭВМ третьего поколения были машины американской фирмы IBM. В Советском Союзе выпускали аналогичные им ЭВМ серии ЕС.

Дальнейший прогресс состоял прежде всего в отработке технологии размещения все большего и большего числа вентилей в одной микросхеме. Это число называется *степенью интеграции*. Для первых интегральных схем оно было невелико (единицы и десятки вентилей). Постепенно степень интеграции увеличивалась и сейчас достигает миллионов вентилей.

Вехой на пути совершенствования интегральных схем стал момент, когда удалось сделать в одной микросхеме целый процессор ЭВМ. Появление в 1971 г. таких процессоров, названных ***микропроцессорами***, ознаменовало переход к *четвертому поколению* ЭВМ. В результате ЭВМ стали дешевле и надежнее.

24.13. Изготовление микросхем

При создании микросхем отнюдь не приходится последовательно изготовлять каждый транзистор. С помощью процесса, похожего на печать фотокарточек, в результате одной серии технологических операций на полупроводниковой пластине одновременно формируют сотни тысяч нужным образом соединенных вентилей. Другими словами, примеси нужных веществ, слои металла и изолятора наносятся одновременно во все необходимые места пластинки.

Более того, в процессе «печати» примеси наносятся на поверхность не одной микросхемы, а большой пластины (диаметром около 10 см), которая после обработки распиливается на сотни отдельных микросхем.

Таким образом, миллионы транзисторов, соединенных в сотни сложных микросхем, изготавливаются всего за десяток операций, в ходе которых на поверхность пластины наносятся разные вещества.

Размеры деталей современных микросхем составляют несколько микрон (в десятки раз меньше толщины человеческого волоса). Поэтому мельчайшие загрязнения в процессе производства (например, пылинки в воздухе или даже водяные пары от дыхания человека) приводят к выпуску бракованных микросхем. Поскольку починить бракованную микросхему невозможно, их приходится выбрасывать. Процент работающих микросхем называется *выходом годных* и может изменяться от 1—2 до 95%. Процент выхода годных микросхем зависит от их сложности и типа и дает представление о культуре и технологии производства.

ЗАДАЧИ И УПРАЖНЕНИЯ

1. Нарисуйте схему вентиля «и-не» из МОП-транзисторов, соответствующую формуле выход = **не** (вход1 **и** вход2).
2. Нарисуйте схему из вентилей «или-не», соответствующую формуле выход = **не** (вход1 **и** вход2). Сравните число транзисторов в этой схеме с числом транзисторов в вашем решении упражнения 1.
3. Нарисуйте схему из вентилей, изображенных на рисунке 94, соответствующую формуле:
   а) выход = (вход1 > = вход2);
   б) выход = (вход1 = вход2);
   в) выход = (вход1 **и** вход2 **и** вход3).

4. Нарисуйте схему только из вентилей «и-не», соответствующую формуле:
   а) выход = **не** вход;
   б) выход = вход1 **и** вход2;
   в) выход = вход1 **или** вход2;
   г) выход = (вход1 = вход2).
5. Решите упражнение 4, используя только вентили «или-не».
6. Нарисуйте схему двухразрядного сумматора.
7. Замените в схеме триггера (рис. 99) вентили «или-не» на вентили «и-не». Опишите работу получившегося триггера.
8. Считая, что электронная лампа (аналог транзистора) сравнима по размерам с обычной осветительной лампочкой, оцените, какой размер имел бы 1 мегабайт памяти в ЭВМ первого поколения.

25 Работа процессора

Компьютер может выполнять сложные алгоритмы, перерабатывая огромные объемы информации. Эта сложнейшая работа состоит из громадного числа «электронных переключений», которые выполняются в компьютере в соответствии с заданной программой. В этом параграфе мы проанализируем работу компьютера на более высоком уровне: познакомимся с тем, что такое машинная программа, какие элементарные команды (операции) умеет выполнять компьютер и, главное, как обеспечивается автоматизм работы процессора. Если сравнить транзисторы и триггеры с песчинками в стенах здания компьютера, то теперь мы познакомимся с кирпичиками, из которых оно сложено.

25.1. Память, процессор, программа

Напомним, что информация в памяти компьютера кодируется последовательностями нулей и единиц, т. е. с помощью бит. Группа из 8 бит называется байтом. Память компьютера в целом можно представлять себе как линейную таблицу байт. Байты в памяти нумеруются с нуля: 0, 1, 2, 3, ... Номер байта называется его ***адресом***.

Выполняемая компьютером ***машинная программа***, как и любая другая информация, кодируется некоторой последовательностью байт и размещается в памяти. Адрес первого байта программы называется ***адресом программы***. Программа состоит

из ***машинных команд*** (или просто команд). Как мы уже говорили, компьютер выполняет лишь очень простые команды: сложение двух чисел, сравнение чисел и т. п.

Важнейшее устройство компьютера — ***процессор***. Именно процессор команда за командой выполняет программы. Процессор имеет небольшую собственную память. Часть этой памяти называется ***счетчиком команд*** (или просто СК) и содержит неотрицательное целое число — адрес следующей машинной команды, которую будет выполнять процессор. Еще одна часть собственной памяти процессора называется ***слово состояния процессора*** (ССП). В частности, при выполнении всех команд сложения, вычитания, сравнения и пр. в ССП запоминается знак *S* результата команды:

если результат меньше нуля, то S = –1,
если результат равен нулю, то S = 0,
если результат больше нуля, то S = +1.

25.2. Основной алгоритм работы процессора

Процессор начинает работу после того, как программа помещена в память компьютера, а в счетчик команд записан адрес первой команды программы. Работу процессора можно описать следующим циклом:

```
нц
| чтение команды из памяти по адресу, записанному в СК
| увеличение СК на длину прочитанной команды
| выполнение прочитанной команды
кц
```

Обратите внимание, что после чтения очередной команды процессор увеличивает СК на длину команды. Поэтому при следующем выполнении тела цикла процессор прочтет и выполнит следующую команду программы, потом еще одну и т. д. Цикл закончится, когда встретится и будет выполнена специальная команда стоп. В итоге компьютер автоматически, без участия человека, команда за командой выполнит *всю программу* целиком.

Автоматизм работы процессора, возможность выполнения длинных последовательностей команд без участия человека — одна из основных отличительных особенностей компьютера как универсальной машины обработки информации.

25.3. Примеры команд процессора

Поясним принцип работы процессора на примере нескольких команд школьного компьютера «Электроника УКНЦ» (табл. 19). Сейчас этот компьютер считается устаревшим, но его команды очень удобны для объяснения, а общий принцип действия даже самых современных компьютеров остается тем же, что был еще у ЭВМ первого поколения.

Каждая команда в «Электронике УКНЦ» занимает в памяти 1, 2 или 3 слова (***словом*** называется пара байт с адресами 0—1, 2—3, 4—5 и т. д.). Целое число также занимает в памяти компьютера 1 слово.

ТАБЛИЦА 19. **Система команд «Электроники УКНЦ»**

Команда	Длина в словах	Что происходит при выполнении команды
стоп	1	Окончание работы
очистить x	2	x := 0
увеличить x на единицу	2	x := x + 1
уменьшить x на единицу	2	x := x – 1
переслать из x в y	3	y := x
прибавить x к y	3	y := y + x
вычесть x из y	3	y := y – x
сравнить x с y	3	S := sign (x – y)
сравнить x с нулем	2	S := sign (x)
если меньше, переход на d	1	Проверяется знак S в ССП Если указанное в команде условие выполнено, то СК := СК + 2*d иначе СК не меняется
если равно, переход на d	1	
если больше, переход на d	1	
если не меньше, переход на d	1	
если не равно, переход на d	1	
если не больше, переход на d	1	
переход на d	1	

Конечно, мы приводим лишь малую толику команд компьютера «Электроника УКНЦ». И даже для этих команд рассматриваем только один простейший вариант (в то время как некоторые команды имеют свыше 100 различных модификаций!). Тем не менее это реальные команды реального компьютера.

25.4. Пример простейшей машинной программы

Пусть в памяти по адресам 100 и 102 расположены два целых числа. Требуется вычислить сумму чисел и поместить ее в память по адресу 104. Это можно сделать с помощью двух команд:

переслать из 100 в 104
добавить 102 к 104

Чтобы заставить процессор выполнить эти две команды, их надо поместить в память, например начиная с адреса 0, поместить вслед за ними команду стоп и записать в СК адрес первой команды, т. е. 0. Предположим, что это сделано и что в начальный момент в памяти по адресу 100 записано число 11, по адресу 102 — число 15, а по адресу 104 — число 7 (рис. 100).

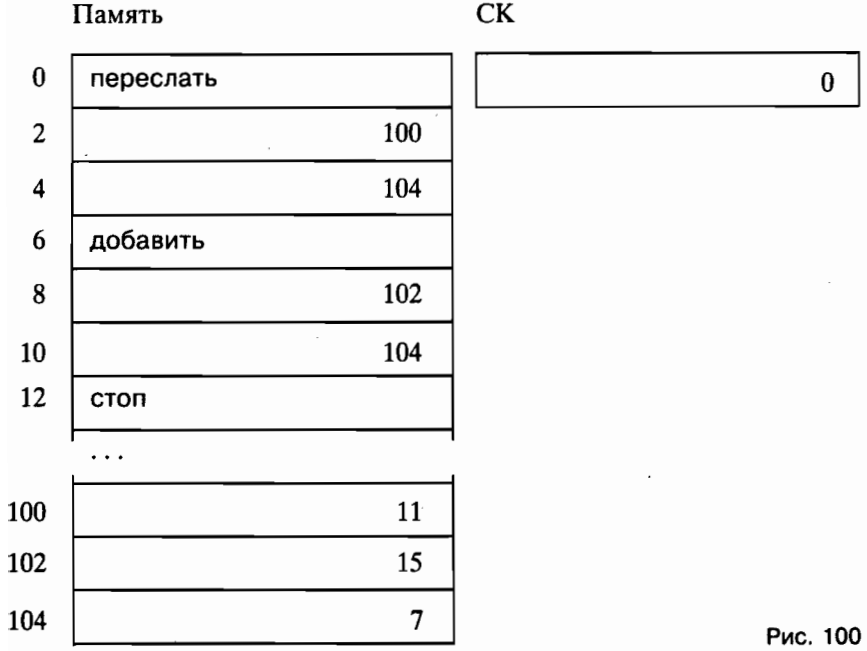


Рис. 100

Состояние компьютера после выполнения этой программы показано на рисунке 101.

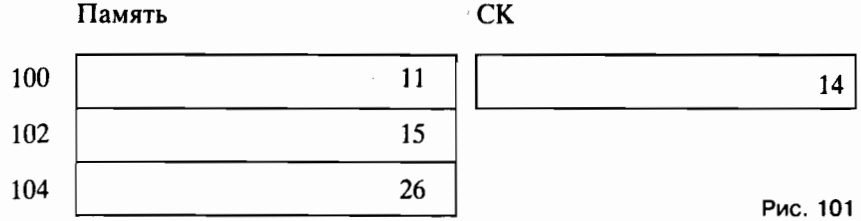


Рис. 101

И команды и числа изображены на рисунках в привычном для человека виде. Напомним, что в компьютере они кодируются последовательностями из 0 и 1. Как именно, нам не важно, но для полноты картины мы один раз такую кодировку покажем (табл. 20).

ТАБЛИЦА 20. **Кодирование реальной программы УКНЦ**

Адрес	Содержимое памяти	Команда
0	0 001 111 111 111 111	переслать
2	0 000 000 001 100 100	100
4	0 000 000 001 101 000	104
6	0 110 111 111 111 111	добавить
8	0 000 000 001 100 110	102
10	0 000 000 001 101 000	104
12	0 000 000 000 000 000	стоп

25.5. Команды условного и безусловного переходов

Среди команд процессора есть такие, которые изменяют значение счетчика команд. Такие команды называются командами ***перехода***. Команды переходов меняют или не меняют СК в зависимости от величины S, запомненной в ССП. Например, команда если меньше переход на d выполняется так:

если S = –1 **то** СК := СК + 2*d **все**

Заметим, что процессор выполняет команду после увеличения СК на длину команды. Поэтому, если условие не выполнено,

процессор просто перейдет к выполнению следующей команды. Если же условие выполнено, то процессор либо пропустит d слов ($2*d$ байт) в памяти (при положительном d), либо вернется на d слов назад (при отрицательном d).

Существует также команда ***безусловного перехода*** переход на d, которая увеличивает СК на $2*d$ независимо ни от чего.

25.6. Машинная реализация цикла пока

Уменьшив содержимое СК, можно заставить процессор повторить некоторую последовательность команд (и таким образом реализовать цикл). Рассмотрим, например, машинную программу, которая помещает в память по адресу 102 сумму чисел $K + (K-1) + ... + 2 + 1$, где K — число, записанное по адресу 100 (рис. 102).

Здесь и в упражнениях содержимое слова памяти по соответствующему адресу обозначено С100, С102, С104 и др. В приведенной программе С100 является аргументом, С102 — результатом, а С104 — промежуточной величиной.

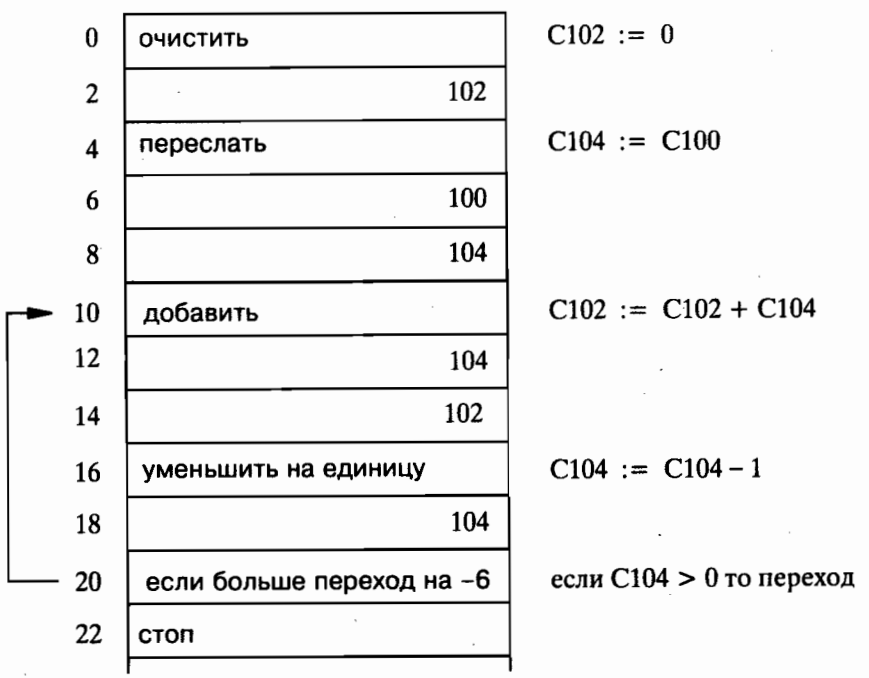


Рис. 102

ЗАДАЧИ И УПРАЖНЕНИЯ

1. Напишите машинную программу, при выполнении которой процессор помещает в память по адресу 110:
   а) 0;
   б) 1;
   в) −1;
   г) −C100;
   д) C100 + C102 + C104;
   е) abs(C100);
   ж) max(C100, C102);
   з) количество положительных среди C100, C102;
   и) количество максимальных среди C100, C102;
   к) сумму $2K + (2K - 2) + ... + 4 + 2$, где K = C100;
   л) K-й член последовательности Фибоначчи, где K = C100.

2. В программе пять команд. Может ли число шагов при выполнении этой программы оказаться равным: а) 1; б) 5; в) 100?

3. Напишите программу, которая в C104 и C106 помещает частное и остаток от деления C100 на C102.

4. Напишите программу, которая в C104 помещает наибольший общий делитель чисел C100 и C102.

26 Устройства ввода/вывода информации

В современных компьютерах используется огромное количество самых разнообразных устройств. Мы рассмотрим очень немногие из них, самые простые, чтобы продемонстрировать применение различных физических механизмов передачи информации.

26.1. Клавиатура

Простейшая клавиатура устроена так: под клавишами расположена тонкая полиэтиленовая пленка, склеенная из трех слоев. На верхнем и нижнем слоях нанесены металлизированные полоски (проводники), средний слой служит изолятором. В местах расположения клавиш в среднем слое сделаны небольшие отверстия, поэтому при нажатии на клавишу верхний слой прогибается, касается нижнего и замыкает контакт. Если на i-й проводник нижнего слоя подать «1» (высокое напряжение), то в j-м проводнике верхнего слоя «1» появится только в том случае,

если клавиша на пересечении *i*-го и *j*-го проводников нажата (контакт замкнут). Меняя *i* и анализируя наличие или отсутствие «1» в проводниках верхнего слоя, компьютер может узнать, какую клавишу или группу клавиш нажал человек.

26.2. Монитор

Монитор похож на обычный телевизор. Основной элемент монитора — электронно-лучевая трубка, внутри которой в вакууме расположены отрицательный электрод (электронная пушка) и покрытый специальным веществом — люминофором — экран (положительный электрод). Под действием высоких напряжений (около 25 тыс. вольт) электроны вырываются из пушки, разгоняются в вакууме, ударяются об экран и вызывают свечение люминофора. В зависимости от химического состава люминофоры светятся разными цветами: белым, красным, синим, зеленым, желтым.

С помощью электрических полей вылетевшие из пушки электроны можно ускорить или затормозить, т. е. изменить силу удара об экран и тем самым яркость свечения люминофора.

Как и в телевизоре, вылетевшие из пушки электроны с помощью электрических полей фокусируются так, чтобы все они попали в одну точку (т. е. в каждый момент времени светится только одна точка экрана). Специальная отклоняющая система, однако, заставляет электронный луч быстро (несколько десятков раз в секунду) строчка за строчкой прочерчивать весь экран. Поэтому у человека создается иллюзия видимости целой картинки. Подавая в нужные моменты времени напряжение на систему управления яркостью, можно заставить светиться те или иные точки и создать на экране нужное изображение.

В цветных мониторах экран покрыт точками разных люминофоров с красным, синим и зеленым свечением, а многоцветная картинка получается наложением (совмещением) трех картинок этих базовых цветов.

26.3. Дисковод

Сегодня чаще всего используются дисководы, рассчитанные на гибкие магнитные диски диаметром 3,5 дюйма (~ 89 мм; 1 дюйм равен 25,4 мм).

Гибкий магнитный диск, или ***дискета*** (рис. 103), представляет собой круг из пластика, покрытый с обеих сторон магнитным слоем. Этот слой похож на покрытие магнитной ленты для обыкновенных магнитофонов. В дисководе, как и в магнитофоне, запись делается с помощью магнитной головки. Только записывается информация не вдоль ленты, а по невидимым кольцевым дорожкам на вращающемся диске. В отличие от магнитной ленты, которую нередко приходится подолгу перематывать, магнитную головку в дисководе сразу устанавливают на нужную дорожку. Гибкие диски, как и кассеты в магнитофоне, являются сменными — можно вставлять в дисковод разные диски, а один и тот же диск использовать в разных дисководах на разных ЭВМ.

26.4. Принтер

Не так давно самыми распространенными были так называемые точечно-матричные принтеры. Главная деталь таких принтеров — печатающая головка с тонкими иголочками. Каждую иголочку выдвигает вперед свой электромагнит, когда по его катушке проходит импульс тока. В головке имеется 9 или 24 расположенных вертикально, одна над другой, иголочек. Выдвигаясь, иголочки ударяют по красящей ленте и оставляют на бумаге точку. При печати головка мелкими шажками движется слева направо, после каждого шага печатая очередную колонку точек. На рисунке 104 показано, как из таких точек получается буква К.

Сейчас матричные принтеры выходят из употребления, их место занимают ***лазерные*** и ***струйные***.

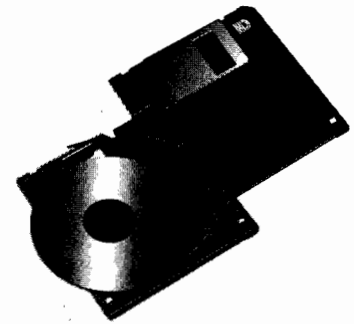


Рис. 103

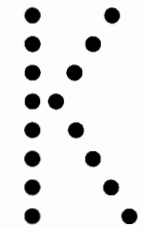


Рис. 104

26.5. Взаимодействие основных частей компьютера. Магистраль

Как же связаны между собой процессор, память и устройства ввода/вывода? В современной электронике для этого чаще всего используется ***магистраль***. Магистраль можно представлять как пучок проводов, к которому подключены параллельно все компоненты компьютера (рис. 105). Магистраль содержит шину адреса, шину данных и некоторые дополнительные провода (например, для указания того, надо ли данные по некоторому адресу записать или прочесть). Посылая по магистрали электрические сигналы, компоненты компьютера могут передавать информацию друг другу.

Как процессор управляет устройствами? При подключении через магистраль любая команда устройству выглядит как команда на запись или чтение некоторой информации по специальному адресу. В «Электронике УКНЦ», например, адрес может меняться в диапазоне от 0 до 65 535. Но только адреса от 0 до 57 343 относятся к памяти, а остальные распределены между разными устройствами (часть адресов не задействована вовсе). Так, устройству печати (принтеру) выделены адреса 65 356 и 65 358, клавиатуре — 65 392 и 65 394.

Вызов команды принтера выглядит так: процессор посылает по магистрали код адреса 65 356, код команды и сигнал «запись». Каждое подключенное к магистрали устройство анализирует код адреса. Принтер обнаруживает «свой» адрес и включает схемы, которые по сигналу «запись» и состоянию шины данных выполняют соответствующую команду. Посылая разную информацию, можно заставить принтер выполнять разные команды.

Аналогичным образом по сигналу «чтение» и коду адреса 65 394 клавиатура посылает по шине данных информацию о своем состоянии, проанализировав которую можно узнать, какую клавишу нажал человек.

Рис. 105

Напомним, что слова «посылает», «анализирует», «обнаруживает» и пр. означают просто срабатывание соответствующих схем и вентилей, появление в проводниках высокого (+5 В) или низкого (0 В) напряжения.

Таким образом, для управления устройствами в УКНЦ нет никаких специальных команд — применяются уже известные нам команды переслать из x в y, сравнить x с y и др. (в которых указываются выделенные устройству адреса). А разные устройства отличаются друг от друга только тем, какие именно адреса им выделены и как (какими наборами 0 и 1) кодируются их команды и состояния.

В современных персональных компьютерах магистраль называют ***шиной*** и для работы с внешними устройствами и с памятью используются разные шины, но общий принцип остается тем же самым.

26.6. Устройства и исполнители

С логической точки зрения все компоненты компьютера, кроме процессора, — самые обычные исполнители. Память компьютера, например, — это исполнитель с двумя командами: прочесть/записать слово по заданному адресу. Лишь процессор играет особую роль: автоматически выполняя программу, он командует всеми остальными устройствами.

Для управления устройствами не обязательно писать машинные программы. Поскольку устройства отличаются от *Робота* и *Чертежника* только системой команд, то для управления ими можно составлять алгоритмы и на алгоритмическом языке. Покажем, как это делается, на примере монитора и клавиатуры.

26.7. Исполнитель *Монитор* (Экран)

С логической точки зрения удобно считать, что монитор — это исполнитель, который может окрасить в любой цвет любую точку экрана, а кроме того, умеет устанавливать курсор в заданную позицию экрана и выводить в эту позицию символ. На алгоритмическом языке эти команды записывают так:

цвет (**арг цел** n)	выбор текущего цвета или яркости
точка (**арг цел** i, j)	окраска точки (i, j) в текущий цвет
поз (**арг цел** y, x)	установка курсора в позицию x строки y
вывсим (**арг сим** s)	вывод символа s в позицию курсора (курсор смещается на символ вправо)

При выполнении команды **вывод** компьютер преобразует тексты и значения величин в строку символов и вызывает соответствующую последовательность команд вывсим. Встретив служебное слово **нс**, компьютер вызывает команду поз, указывая в ней координаты начала следующей строки, и т. п.

26.8. Исполнитель *Клавиатура*

Каждая клавиша на клавиатуре имеет свой числовой код. Компьютер постоянно анализирует состояние клавиатуры, определяет код нажатой клавиши и выполняет соответствующие этой клавише действия.

Например, при выполнении команды **ввод** компьютер последовательно запрашивает у клавиатуры коды клавиш до тех пор, пока человек не нажмет клавишу Enter. В процессе ввода компьютер вызывает команды вывсим для отображения вводимых символов на экране. По окончании ввода компьютер анализирует полученную строку символов и присваивает указанным в команде **ввод** величинам соответствующие значения.

В алгоритмическом языке можно явно запросить у клавиатуры код нажатой клавиши с помощью команды

клав (**рез цел** k)

При выполнении вызова "клав (k)" компьютер запрашивает у клавиатуры код клавиши (либо ждет, пока человек нажмет какую-нибудь клавишу, если она еще не нажата). Код нажатой клавиши (целое число от 0 до 255) присваивается величине *k*. Изображение на экране монитора при этом не меняется.

ЗАДАЧИ И УПРАЖНЕНИЯ

1. Для подключения *Робота* к УКНЦ выделены два адреса: 65 352 и 65 354. Придумайте способ кодировки команд *Робота* и передаваемой им информации. Используя эту кодировку, переведите в машинную программу алгоритмы А1, А20, А28, А56.

2. Опишите, как будет выполняться следующий алгоритм:

А 108

```
алг квадраты
нач цел n, k
| n := 0; k := 0
| нц пока k <> 13 | 13 — код клавиши Enter
| | n := n + 1
| | поз (1, 1); вывод "n = ", n, ", n*n = ", n*n
| | клав (k)
| кц
кон
```

3. Составьте алгоритм, который очищает (заполняет пробелами) весь экран и устанавливает курсор в его левый верхний угол.

4. Составьте алгоритм, который выводит строку символов **лит** t на экран вертикально: сверху вниз, начиная с позиции курсора.

5. Сопоставим первым пяти командам *Робота* по одной букве:

Команда	вверх	вниз	влево	вправо	закрасить
Буква	В	Н	Л	П	К

По образцу упражнения 2 составьте алгоритм, который при нажатии на одну из клавиш «В», «Н», «Л», «П», «К» выдает соответствующую команду *Роботу*, потом ждет нажатия на следующую клавишу, опять выдает команду *Роботу* и т. д. до тех пор, пока не будет нажата клавиша Enter.

6. Аналогично упражнению 5 составьте алгоритм, который при нажатии на клавиши «В», «Н», «Л», «П» рисует с помощью *Чертежника* отрезок единичной длины в соответствующем направлении.

7. Составьте алгоритм, который с помощью команды точка рисует:
   а) горизонтальный;
   б) вертикальный;
   в) диагональный;
   г) произвольный отрезок с заданными концами.

8. Составьте алгоритм, который с помощью команды точка рисует: а) прямоугольник; б) окружность; в) круг.

9. Решите упражнение 6, используя вместо *Чертежника* вспомогательные алгоритмы из упражнений 7, а), б) (решение этой задачи можно назвать простейшим графическим редактором).

27 Работа компьютера

Устройство и работу компьютера можно изучать на разных уровнях — интересоваться физическими основами электронной обработки информации (§ 24) или, забыв про напряжение, транзисторы, вентили, микросхемы и пр., изучать логическое устройство процессора: систему его команд, алгоритм работы, структуру хранимой во внутренней памяти информации (§ 25). В этом параграфе мы поднимемся еще на один уровень и попробуем описать работу компьютера в целом.

27.1. Алгоритмический язык и машинные коды

Компьютер может выполнять программы, записанные на разных языках программирования: на Бейсике, Паскале, школьном алгоритмическом языке и многих других. В § 25 мы познакомились с программами в машинных кодах и теперь знаем, что фактически компьютер всегда работает по такой — ***машинной*** — программе. Но программа на алгоритмическом языке и программа в машинных кодах — это, как говорится, две большие разницы. Возникает вопрос: как же компьютер выполняет программы, записанные на алгоритмическом или ином языке программирования?

Существует два принципиально разных способа выполнения таких программ. Они называются компиляцией и интерпретацией.

27.2. Компиляция

Программа на алгоритмическом языке — это информация, а компьютер — универсальная машина для ее обработки. Поэтому можно составить машинную программу (она называется ***компилятором***), при выполнении которой компьютер обработает ***текст*** программы на алгоритмическом языке и преобразует его в эквивалентную программу в машинных кодах (т. е. в последовательность байт). Само такое преобразование называется ***компиляцией***. Поскольку в результате компиляции получается программа в машинных кодах, то ее уже можно поместить в память компьютера и выполнить.

Таким образом, при компиляции работа происходит в два этапа: на первом этапе компьютер переводит программу с алгоритмического языка в машинные коды, а на втором — выполняет полученную программу в машинных кодах.

27.3. Интерпретация

Можно составить другую программу в машинных кодах (она называется ***интерпретатором***), при выполнении которой компьютер будет анализировать ***текст*** программы на алгоритмическом языке и сразу выполнять предписанные этой программой действия, не переводя ее в машинные коды. Такой способ выполнения программы называется ***интерпретацией***.

27.4. Компиляция и интерпретация

Разницу между компиляцией и интерпретацией можно пояснить с помощью аналогии. Пусть требуется приготовить пирог по рецепту, написанному на иностранном языке. Есть два пути. Можно сначала перевести (скомпилировать) рецепт на родной язык и лишь затем готовить пирог по рецепту на родном языке. Можно, однако, по мере чтения и перевода рецепта сразу выполнять требуемые им действия (это соответствует интерпретации). В последнем случае мы получим не текст рецепта на родном языке, а сразу пирог. Правда, если пирог придется готовить несколько раз, рецепт придется переводить многократно.

27.5. Программа начальной загрузки

Чтобы можно было выполнить алгоритм, надо поместить в память компьютера и начать выполнять компилятор или интерпретатор алгоритмического языка. Сам компилятор — это программа в машинных кодах, т. е. последовательность 0 и 1 (информация!). Конечно, процессор может прочесть эту информацию с магнитного диска, но при этом он должен работать по какой-то другой (еще одной!) программе, ведь единственное, что он умеет, — это выполнять машинные программы. Эту еще одну программу тоже можно прочесть с диска или ввести с какого-нибудь из устройств ввода. Но кто это может сделать? Только про-

цессор, а ему понадобится новая программа и т. д. Получается какой-то порочный круг!

Чтобы разобраться в этом вопросе, рассмотрим, что происходит при включении компьютера. Сразу после включения процессор начинает выполнять специальную ***программу начальной загрузки***. Эта программа записывается в постоянную память (ПЗУ) при изготовлении компьютера на заводе и не исчезает при выключении питания. Читать ее с диска не надо.

На большинстве персональных компьютеров выполнение программы начальной загрузки состоит в том, что процессор читает с определенного, заранее фиксированного места на диске другую машинную программу (она называется ***операционной системой***), записывает ее в память и помещает в СК адрес первой команды этой программы (т. е. начинает ее выполнять).

27.6. Операционная система (ОС)

Итак, в результате начальной загрузки в компьютер обычно загружается программа, называемая операционной системой. В отличие от программы начальной загрузки операционная система читается с диска и поэтому может быть изменена.

Что же делает операционная система? Прежде всего, она позволяет человеку выбрать, с какой именно программой он желает дальше работать, т. е. запрашивает у человека имя программы, помещает ее в память и начинает выполнять. Человек может выбрать, например, редактор текстов, компилятор с языка Паскаль или систему программирования КуМир для школьного алгоритмического языка. Дальше компьютер работает уже по новой программе. Так, система КуМир позволяет человеку вводить и выполнять алгоритмы на школьном алгоритмическом языке. Помните: «Дедка за репку, бабка за дедку... мышка за кошку — вытянули репку»? Так и у нас: чтобы выполнить алгоритм («вытянуть репку»), начинать приходится с программы начальной загрузки («мышки»).

Кроме загрузки и выполнения других программ, операционная система обычно дает возможность переписывать информацию с одного диска на другой, распечатывать ее, стирать ненужную информацию, узнавать, сколько на диске незанятого места, и т. п. Все эти действия выполняются по командам человека.

ЗАДАЧИ И УПРАЖНЕНИЯ

1. Какая информация хранится в памяти компьютера при интерпретации алгоритма на алгоритмическом языке?

2. Опишите результат выполнения алгоритма

А 109

```
алг интерпретатор (арг лит t)
    дано | строка t состоит только из букв "В", "Н", "Л", "П", "К"
    надо | Робот выполнил соответствующую последовательность
         | команд (соответствие задано в упражнении 5 § 26)
нач цел i
| нц для i от 1 до длин (t)
| | выбор
| | | при t[i] = "В" : вверх
| | | при t[i] = "Н" : вниз
| | | при t[i] = "Л" : влево
| | | при t[i] = "П" : вправо
| | | при t[i] = "К" : закрасить
| | все
| кц
кон
```

при вызове с аргументом «КВВКНЛКППК». Объясните, почему алгоритм называется интерпретатором.

3. Составьте алгоритм:

```
алг компилятор (арг лит t)
    дано | строка t состоит только из букв "В", "Н", "Л", "П", "К"
    надо | на экран выведена машинная программа (столбик
         | слов, где каждое слово изображено последователь-
         | ностью 0 и 1), которая содержит вызовы соответст-
         | вующих команд Робота (используйте ваше решение
         | упражнения 1 § 25)
```

Объясните, почему алгоритм называется компилятором.

4. Составьте алгоритм:

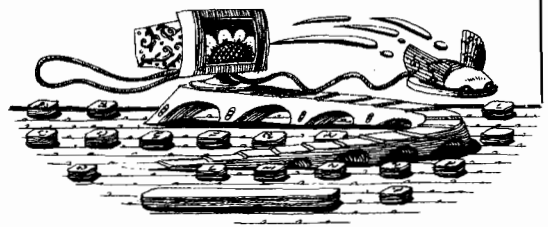
```
алг компилятор (арг лит t)
    дано | строка t состоит только из букв "В", "Н", "Л", "П", "К"
    надо | на экран выведен текст алгоритма на алгоритмиче-
         | ском языке, содержащий соответствующую последо-
         | вательность вызовов команд Робота
```

Объясните, почему алгоритм называется компилятором.

5. На некоторых компьютерах операционная система размещена не на диске, а в ПЗУ. Опишите достоинства и недостатки такого подхода.

ГЛАВА 5

Информационные модели



28 Информационные модели

28.1. Компьютер обрабатывает формализованную информацию

Как мы уже знаем, компьютер — это универсальная информационная машина, его основное (и единственное) предназначение — обработка различной информации.

Универсальность компьютера заключается в том, что с его помощью можно обрабатывать самую разнообразную информацию. Современные компьютеры работают не только с числами и текстами, но и с изображениями, со звуками, к ним можно подключать различные приборы и т. д.

И все же нельзя сказать, что компьютер может работать с *любой* информацией. Прежде чем обрабатывать информацию, ее надо *ввести* в компьютер, а для этого необходимо представить в форме, пригодной для компьютерной обработки. Эта форма может меняться в зависимости от того, какими средствами мы собираемся пользоваться, но в любом случае информация должна быть четко описана по определенным правилам.

Например, при создании баз данных необходимо задать структуру хранимой информации по правилам используемой

СУБД, а при разработке алгоритмов информация описывается в виде величин.

Строго описанная информация называется ***формализованной***. Компьютер обрабатывает только ***формализованную информацию***.

28.2. Задача формализации — построение информационной модели

В реальных задачах обычно фигурируют не величины, а объекты и процессы. Но в компьютер нельзя ввести ни объект, ни процесс, он может обрабатывать только информацию о них.

Следовательно, для решения реальной задачи с помощью компьютера необходимо проанализировать объект или процесс, определить, какую информацию о нем мы хотим обрабатывать, и формализовать ее.

При этом происходит замена реального объекта или процесса информацией о нем. Общее описание этой информации, ее структура называется ***информационной моделью***. Например, при проектировании баз данных мы описывали структуры записи. Это описание можно рассматривать как пример информационной модели.

При использовании различных компьютерных средств применяются разные способы построения информационной модели. В этой главе мы рассмотрим создание информационных моделей с помощью алгоритмического языка.

28.3. Информационная модель — набор величин

Как мы уже знаем, при использовании алгоритмического языка вся информация, которую обрабатывает алгоритм, должна быть представлена в виде величин.

Таким образом, построение информационной модели сводится к представлению информации об объекте или процессе в виде набора величин. Эти величины и есть информационная модель.

28.4. Простейший пример информационной модели

Многие из вас, вероятно, видели кассира, продающего билеты в кино. Обычно у него есть план зрительного зала, на котором он отмечает уже проданные места. Этот план можно считать моделью кинозала, только некомпьютерной.

Построим информационную модель кинозала средствами алгоритмического языка.

Предположим, что в зале ровно 28 рядов, а в каждом из них 20 мест. Тогда информацию о проданных билетах удобно хранить в прямоугольной таблице:

М 1

```
лог таб продано [1:28, 1:20] | продано [i, j] = да <=>
                             | место j ряда i продано
```

Эту таблицу можно считать простейшей информационной моделью кинозала, предназначенной для продажи билетов. С ее помощью легко, например, подсчитать, сколько всего билетов продано:

```
n := 0
нц для i от 1 до 28          | для каждого ряда
| нц для j от 1 до 20        | для каждого места в ряду
| | если продано [i,j]       | если место продано
| | | то n := n + 1          | то учесть его
| | все
| кц
кц
```

Примечание. В этом параграфе мы приводим не полные алгоритмы с заголовками и описаниями, а только их фрагменты. Причины этого будут объяснены в следующем параграфе.

Построенная модель позволяет решать и некоторые другие задачи, например находить свободные места как можно ближе к экрану или подальше от него.

А вот определить выручку от продажи билетов с помощью этой модели нельзя — в ней нет информации о цене билетов.

Дополним модель, включив в нее сведения о цене. Если цена всех мест одинакова, это делается очень просто:

М 2

```
лог таб продано [1:28, 1:20] | продано [i, j] = да <=> место j ряда i
                             | продано
цел цена                     | цена в рублях
```

Для определения выручки в этой модели достаточно подсчитать количество проданных билетов (это мы уже сделали) и умножить его на цену.

В некоторых кинотеатрах цена билета изменяется в зависимости от ряда. Чтобы учесть эту информацию, надо еще раз изменить модель, сделав цену не простой величиной, а таблицей:

М 3

```
лог таб продано [1:28, 1:20] | продано [i,j] = да <=>место j ряда i
                             | продано
цел таб цена [1:28]          | цена [i] = цена билета в i-м ряду
```

Теперь подсчет выручки можно сделать так:

```
s := 0
нц для i от 1 до 28          | для каждого ряда
| нц для j от 1 до 20        | для каждого места в ряду
| | если продано [i,j]       | если место продано
| | | то s := s + цена [i]   | то учесть его цену
| | все
| кц
кц
```

28.5. Одному объекту могут соответствовать разные модели

Информационная модель, как и любая другая, включает в себя далеко не все свойства реальных объектов. Например, построенная нами модель никак не отражает свойств кинозала как помещения: в модели нет ни размеров зала, ни высоты, ни расположения дверей. Дело в том, что вся эта информация не нужна для решения интересующей нас задачи — продажи билетов.

А вот если бы мы решали задачи, связанные с архитектурным проектированием или пожарной безопасностью кинозала, нас бы в первую очередь заинтересовала именно эта информация, и она легла бы в основу соответствующей информационной модели. Но это была бы уже совсем другая модель того же объекта.

Таким образом, одному и тому же объекту могут соответствовать разные модели. Выбор модели определяется целью, ради которой эта модель составляется, т. е. задачами, которые мы собираемся решать.

Построение информационной модели в реальных задачах требует серьезных знаний в той области, к которой относится задача. Именно при создании модели информатика стыкуется с другими науками.

28.6. Формализация тоже может быть неоднозначной

После того как выделена необходимая информация (в реальных задачах это делается с участием экспертов — специалистов в соответствующей предметной области), ее необходимо формализовать, т. е. описать по строгим правилам, например в виде набора величин алгоритмического языка.

Одну и ту же информацию часто можно формализовать разными способами. Так, в модели кинозала можно отмечать проданные места как элементы таблицы не с логическими, а с целыми значениями:

M 4

```
цел таб продано [1:28, 1:20] | продано [i, j] = 1 <=>
                             | место j ряда i продано
                             | продано [i,j] = 0 <=>
                             | место j ряда i свободно
цел таб цена [1:28]          | цена [i] = цена билета в i-м ряду
```

В этом случае необходимо изменить алгоритмы работы с моделью. Например, число проданных билетов можно узнать, просто сложив все элементы таблицы продано:

```
n := 0
нц для i от 1 до 28
|  нц для j от 1 до 20
|  |  n := n + продано [i, j]
|  кц
кц
```

28.7. Обобщение модели

Наша модель кинозала рассчитана на зал с конкретными размерами. Но ведь для алгоритмов обработки не так уж важно точное количество рядов и мест. Достаточно заменить числовые константы в модели и алгоритмах, и мы получим модель другого зала и алгоритмы ее обработки.

Возникает естественное предложение — обобщить модель, сделав количество рядов и мест переменными. Это нетрудно сделать, используя таблицы с переменными размерами. Но при обработке информации о конкретном зале его реальные размеры необходимо знать, следовательно, эту информацию тоже надо включить в модель.

M 5

```
цел чр, чм; чр :=28; чм :=20 | чр — число рядов, чм — число мест
лог таб продано [1:чр, 1:чм] | продано [i, j] = да <=>
                             | место j ряда i продано
цел таб цена [1:чр]          | цена [1] = цена билета в ряду i
```

28.8. Постепенное расширение модели

После построения модели можно составлять различные алгоритмы, ее использующие. Иногда при этом выясняется, что для решения каких-то задач информации недостаточно. Тогда приходится уточнять модель и вносить в нее дополнительную информацию.

Рассмотрим пример. Попробуем с помощью модели кинозала найти два свободных места рядом.

```
лог нашли, цел ряд, место
нашли := нет
ряд := 1
нц пока не нашли и ряд <= чр
|  место := 1
|  нц пока не нашли и место < чм
|  |  если не продано [ряд, место] и не продано [ряд, место+1] | *
|  |  |  то нашли := да
|  |  |  вывод нс,"Соседние места",место,"и",место+1,"в",ряд, "ряду"
|  |  все
|  |  место :=место+1
|  кц
|  ряд :=ряд+1
кц
если не нашли
|  то вывод нс, "Соседних свободных мест нет"
все
```

В этом фрагменте предполагается, что все ряды «непрерывны», т. е. в зале нет проходов, перпендикулярных экрану. Однако в реальных залах такие проходы обычно есть, а для решения поставленной задачи их наличие весьма существенно. Согласи-

тесь, что не очень приятно купить билеты на кажущиеся соседними места, а потом оказаться через проход друг от друга. У реальных кассиров на плане зала обычно отмечены все проходы, и эта проблема не возникает.

Попробуем учесть в модели информацию о проходах. Предположим, что в зале есть один проход, который разделяет одни и те же места в каждом ряду. Добавим в модель информацию об этом проходе:

```
цел проход | места проход и проход+1 в каждом ряду
           | разделены
```

Учесть эту информацию в алгоритме совсем просто. Надо каждый раз проверять, не разделены ли найденные места проходом. Для этого достаточно заменить строку, помеченную звездочкой, на такую:

```
если место <> проход и не продано [ряд, место]
и не продано [ряд, место+1]
```

28.9. Еще один пример информационной модели

Пусть в некоторой стране 100 городов и между некоторыми из них летают самолеты.

Построим информационную модель, позволяющую отвечать на вопросы, как перелететь из одного города в другой, каково минимальное число пересадок и какова минимальная цена перелета. Модель будем строить сразу в обобщенном виде — для произвольного числа городов.

М 6

```
цел чг; чг :=100          | число городов
лит таб назв [1:чг]       | назв [i] — название i-го города
лог таб рейс [1:чг, 1:чг] | рейс [i, j] = да <==> между городами
                          | i и j есть прямой авиарейс
цел таб цена [1:чг, 1:чг] | цена [i, j] = цена перелета из i в j
```

С помощью модели М6 можно, например, заставить компьютер проанализировать, как добраться из города $i1$ в город $i2$, совершив не более одной пересадки (ниже используются промежуточные величины **цел** р, **лог** найден пункт пересадки):


```
вывод "Из города", назв [i1], "в город", назв [i2], нс
если рейс [i1, i2] то вывод "есть прямой рейс"
| иначе
| р := 0; найден пункт пересадки := нет
| нц пока р < чг и не найден пункт пересадки
| | р := р + 1
| | найден пункт пересадки := рейс [i1, р] и рейс [р, i2]
| кц
| если найден пункт пересадки
| | то вывод "есть маршрут с одной пересадкой в", назв [р]
| | иначе вывод "с одной пересадкой добраться нельзя"
| все
все
```

28.10. Избыточность информационных моделей

В модели сети авиалиний в таблице *цена* хранится цена только для существующих рейсов, т. е. цена $[i, j]$ имеет смысл только для таких i и j, для которых рейс [i, j] = **да**.

Элементы таблицы *цена*, которым не соответствуют никакие рейсы, не стоит оставлять неопределенными, так как это может привести к ошибкам. Обычно в подобных случаях такие элементы заполняют каким-нибудь «невозможным» значением. Например, в качестве «невозможной» цены можно взять значение –1.

В этом случае таблица *рейс*, вообще говоря, становится необязательной, так как всю информацию о наличии тех или иных рейсов можно получить из таблицы *цена*.

Ситуация, когда одна и та же содержательная информация хранится в разных величинах модели, называется ***информационной избыточностью***.

Иногда избыточности стараются избегать. Дело в том, что при наличии избыточности модель при некоторых значениях величин может стать ***противоречивой***. Например, если в таблице *рейс* какой-то рейс отмечен как существующий, а в таблице *цена* в соответствующем месте стоит –1, то возникает противоречие, которое свидетельствует об ошибке в исходных данных.

Иногда в модели сознательно заносится избыточная информация, так как при этом получаются более простые и быстрые алгоритмы. Хорошей иллюстрацией этого может служить перевод со словарем. Русско-английский и англо-русский словари со-

держат, вообще говоря, одну и ту же информацию о соответствиях лексических значений слов. Таким образом, при наличии одного из этих словарей второй избыточен — он не содержит новой информации. Но каждый, кто хоть раз переводил какой-нибудь текст, понимает, что перевести с английского на русский, пользуясь русско-английским словарем, практически невозможно.

28.11. Кодирование геометрической информации

Основы числового кодирования геометрической информации заложил великий французский ученый XVII в. Рене Декарт: любую точку плоскости можно задать парой чисел — ее ***декартовыми*** координатами. А от числового кодирования точек нетрудно перейти к кодированию более сложных геометрических объектов, например фигур, известных из школьной планиметрии (табл. 21, рис. 106).

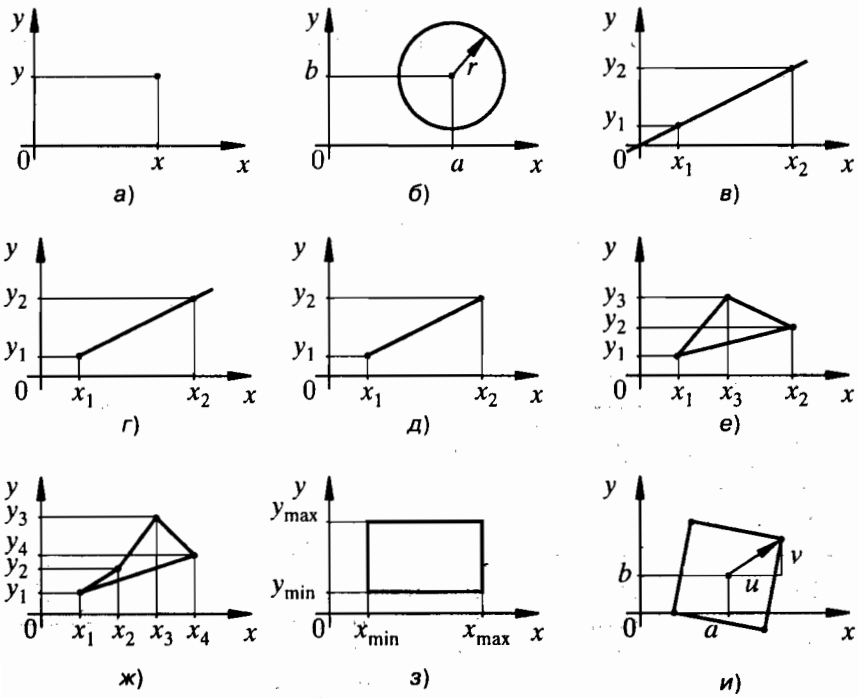


Рис. 106

ТАБЛИЦА 21. **Кодирование геометрической информации**

Геометрическая фигура	Описание		Модель
	на алгоритмическом языке	на геометрическом языке	
Точка	**вещ** x, y	Координаты точки	М 7
Окружность	**вещ** r **вещ** a, b	Радиус окружности, координаты центра	М 8
Прямая	**вещ** x1, y1 **вещ** x2, y2	Координаты двух точек на прямой	М 9
Луч	**вещ** x1, y1 **вещ** x2, y2	Координаты начала луча, координаты точки на луче	М 10
Отрезок	**вещ** x1, y1 **вещ** x2, y2	Координаты начала и конца отрезка	М 11
Треугольник	**вещ** x1, y1 **вещ** x2, y2 **вещ** x3, y3	Координаты вершин треугольника	М 12
n-угольник	**вещ таб** x[1:n], y[1:n]	$(x[i], y[i])$ — координаты *i*-й вершины	М 13
Прямоугольник со сторонами, параллельными осям *Ox*, *Oy*	**вещ** xmin **вещ** xmax **вещ** ymin **вещ** ymax	Точка (x, y) принадлежит прямоугольнику <=> xmin <= *x* <= xmax ymin <= *y* <= ymax	М 14
Квадрат	**вещ** a, b **вещ** u, v	Центр квадрата Координаты вектора из центра к одной из вершин квадрата	М 15

Пользуясь этими моделями, можно переводить геометрические понятия на алгоритмический язык (табл. 22).

ТАБЛИЦА 22. **Примеры применения геометрических моделей**

Геометрический объект	Описание на алгоритмическом языке
Длина отрезка (М11)	(x1 – x2)**2 + (y1 – y2)**2
Точка (М7) внутри окружности (М8)	(x – a)**2 + (y – b)**2 < r**2
Квадрат длины диагонали *n*-угольника (М13)	(x[i] – x[j])**2 + (y[i] – y[j])**2
Периметр *n*-угольника (М13)	p := 0 **нц для** i **от** 1 **до** n – 1 | p := p + длина (i, i + 1) **кц** p := p + длина (n, 1)
Окружность (М8) внутри прямоугольника (М14)	xmin < a – r **и** a + r < xmax **и** ymin < b – r **и** b + r < ymax
Площадь прямоугольника (М14)	(xmax – xmin)*(ymax – ymin)
Площадь квадрата (М15)	2*(u**2 + v**2)

Геометрические примеры наглядно демонстрируют неоднозначность построения информационной модели. Например, отрезок можно задать и так:

М 16

```
вещ a, b | координаты середины отрезка
вещ m    | длина отрезка
вещ ф    | угол наклона отрезка к оси Ox
```

Даже одни и те же данные можно представить разными величинами алгоритмического языка. Например, для задания точки можно использовать таблицу:

М 17

```
вещ таб x[1:2] | точка с координатами (x[1], x[2])
```


28.12. Кодирование алгоритмов управления исполнителями

Вспомните соответствие между первыми пятью командами *Робота* и буквами «В», «Н», «Л», «П», «К» из упражнения 5 § 26. При таком соответствии алгоритм управления *Роботом*, содержащий только эти команды, можно закодировать одной литерной величиной:

М 18

```
лит t | алгоритм для Робота из букв "В", "Н", "Л", "П", "К"
```

Именно эта информационная модель была использована в упражнениях 2—4 § 27.

Рассмотрим еще один пример. Закодируем каждую из команд *Чертежника* одним или тремя вещественными числами (табл. 23).

ТАБЛИЦА 23. **Кодирование команд *Чертежника***

Команда	Код
Поднять перо	1
Опустить перо	2
Сместиться в точку (*x*, *y*)	3 x y
Сместиться на вектор (*a*, *b*)	4 a b

Тогда всякую последовательность команд *Чертежника* можно задать линейной таблицей:

М 19

```
вещ таб a[1:n] | последовательность кодов и аргументов команд
               | Чертежника
```

Например, последовательность команд

поднять перо	1
сместиться в точку (0, 0)	3, 0, 0
опустить перо	2
сместиться на вектор (1.5, –1.5)	4, 1.5, –1.5
поднять перо	1

можно задать таблицей **вещ таб** a[1:9]:

1	3	0	0	2	4	1.5	−1.5	1
1	2	3	4	5	6	7	8	9

Как узнать, какую последовательность команд *Чертежника* кодирует эта таблица? В $a[1]$ записано число 1 — код команды поднять перо. Поскольку эта команда аргументов не имеет, то следующая команда начинается в $a[2]$: $a[2] = 3$ (команда сместиться в точку). Эта команда имеет два аргумента, значит, $a[3]$ и $a[4]$ — ее аргументы, следующая команда начинается в $a[5]$ и т. д.

Выполнить закодированную в таблице a последовательность команд можно с помощью следующего алгоритма:

А 110

```
алг интерпретатор (арг цел n, вещ таб a[1:n])
    дано | таблица a содержит закодированную последователь-
         | ность команд Чертежника (модель М19)
    надо | Чертежник выполнил эту последовательность команд —
         | нарисована соответствующая картинка
нач цел i
| i: = 1
| нц пока i <= n
| | выбор
| | | при a[i]=1 : поднять перо; i :=i+1
| | | при a[i]=2 : опустить перо; i :=i+1
| | | при a[i]=3 : сместиться в точку (a[i+1], a[i+2]); i :=i+3
| | | при a[i]=4 : сместиться на вектор (a[i+1], a[i+2]); i :=i+3
| | все
| кц
кон
```

Величина i в этом алгоритме аналогична счетчику команд, а таблица a — машинной программе (части памяти ЭВМ).

ЗАДАЧИ И УПРАЖНЕНИЯ

1. В рамках модели М5 запишите на алгоритмическом языке: а) начало продажи билетов на новый сеанс (приведение модели в состояние, когда все места свободны); б) продажу билета; в) поиск трех соседних свободных мест.

2. Выполните упражнение 1, в) с учетом наличия в зале одного прохода.
3. Измените модель М5, считая, что места в центре зала стоят дороже, чем места по краям. Запишите на алгоритмическом языке подсчет выручки в измененной модели.
4. Измените модель М5, считая, что места могут бронироваться. Запишите на алгоритмическом языке бронирование и разбронирование мест.
5. Измените модель М5, считая, что в зале проходит 6 сеансов в день: 4 по одной цене и 2 — по другой, а билеты продают только «на завтра».
6. В рамках модели М5 запишите на алгоритмическом языке подсчет выручки, считая, что место номер 13 в 13 ряду продается со скидкой 75%, а все остальные места в 13 ряду, а также все 13-е места в других рядах продаются со скидкой 50%.
7. Составьте информационную модель кинозала, в котором ряды имеют разное количество мест.
8. В рамках модели М6 запишите на алгоритмическом языке фрагмент алгоритма, который выводит на экран:
   а) все маршруты из города i1 в город i2 ровно с одной пересадкой и цену каждого из этих маршрутов;
   б) все маршруты из i1 в i2 с одной пересадкой в порядке возрастания цены маршрута;
   в) самый дешевый маршрут из i1 в i2 с одной пересадкой;
   г) сообщение о том, можно ли добраться из i1 в i2 с двумя пересадками.
9. Линейная таблица **цел таб** p[1:m] содержит номера всех городов, в которые можно добраться из i1, делая не более двух пересадок. В рамках модели М6 запишите фрагмент алгоритма, который выводит на экран сообщение о том, можно ли добраться из i1 в i2, делая не более трех пересадок.
10. Составьте информационную модель линий и станций Московского метро, учитывающую время в пути между соседними станциями и примерное время, которое тратится на пересадки. Переформулируйте упражнение 8 для этой модели и решите его.
11. Закодируйте величинами алгоритмического языка: а) угол; б) пару параллельных прямых; в) произвольно расположенный прямоугольник (все на плоскости).
12. Составьте алгоритм "**алг вещ** S (**арг вещ** x1, y1, x2, y2, x3, y3)", вычисляющий площадь треугольника по формуле Герона. Исполь-

зуя его как вспомогательный, напишите фрагменты алгоритмов, которые вычисляют:

а) высоту треугольника (М12), опущенную из данной вершины;

б) расстояние от точки (М7) до прямой (М9);

в) касается ли окружность (М8) прямой (М9).

13. Напишите формулы перехода от модели отрезка (М11) к модели М16 и обратно.

14. Напишите формулы, выражающие координаты всех четырех вершин квадрата через величины модели М15.

15. Запишите на алгоритмическом языке условие «отрезок (М11) внутри окружности (М8)».

16. Даны два прямоугольника со сторонами, параллельными осям координат (модель М14): *x*min1, *x*max1, *y*min1, *y*max1 и *x*min2, *x*max2, *y*min2, *y*max2. Запишите фрагмент алгоритма, который выводит на экран: а) площадь пересечения; б) периметр объединения этих прямоугольников.

17. Алгоритм управления *Роботом* закодирован строкой **лит** t (модель М18), в которой ровно две буквы «К». Составьте алгоритм, определяющий, сколько клеток (одну или две) закрасит *Робот* при выполнении соответствующего алгоритма.

18. Алгоритм управления *Роботом* закодирован строкой **лит** t (модель М18). Составьте алгоритм, который из **арг лит** t делает строку **рез лит** k, заменяя:

а) каждую группу подряд идущих букв «К» на одну букву «К»;

б) каждую группу от 2 до 9 повторяющихся букв «В», «Н», «П», «Л» на цифру от 2 до 9, за которой следует повторяющаяся буква (при t = "ВВНПППППППППППЛ" должна получиться строка k = "2ВН9П2ПЛ");

в) каждую группу повторяющихся букв «В», «Н», «П», «Л» на число повторений, за которым следует повторяющаяся буква (при t = "ВВНПППППППППППЛ" должна получиться строка k = "2ВН11ПЛ").

19. Составьте алгоритм "**алг** интерпретатор (**арг лит** k)" для выполнения программ, полученных в упражнениях 18, б), в).

20. В строке **лит** t встречаются только буквы «В», «Н», «П», «Л», «К», «Х», а в строке **лит** x — только буквы «В», «Н», «П», «Л», «К». Составьте алгоритм, который:

а) выполняет строку *t*, считая, что буква «Х» кодирует команду вызова вспомогательного алгоритма, закодированного в строке *x*;

б) из строки *t* делает строку *k*, подставляя вместо каждой буквы «Х» содержимое строки *x*.

21. Измените модель М18 так, чтобы она была применима к алгоритмам управления *Роботом*, в которых, кроме команд вверх, вниз, вправо, влево, закрасить, встречаются логические команды *Робота* (сверху стена, клетка закрашена и др.), а также цикл **пока**. Составьте алгоритм-интерпретатор для этой модели.

22. Рассмотрим модель М19 как информационную модель некоторой картинки *X*, состоящей только из отрезков. По образцу алгоритма А110 составьте алгоритм, который с помощью *Чертежника* рисует картинку *Y*:

а) симметричную *X* относительно оси *Ox*;

б) симметричную *X* относительно оси *Oy*;

в) симметричную *X* относительно начала координат;

г) получающуюся из *X* поворотом вокруг начала координат на 90° против часовой стрелки.

23. Придумайте способ кодирования алгоритмов управления *Чертежником* с помощью одной литерной величины **лит** t.

24. Придумайте способ кодирования позиции в игре «крестики-нолики» на доске 15×15 клеток. Как подсчитать количество крестиков на доске? Как узнать, выиграли ли крестики?

25. Придумайте, как закодировать состояние в упражнении 3, а) § 6 (волк, коза и капуста). Как проверить, допустимо ли это состояние (т. е. что никто никого не съест)?

26. Составьте информационную модель расписания уроков: а) одного ученика; б) одного класса (на некоторых уроках класс может делиться на группы); в) всей школы. Как подсчитать число уроков в среду (модели а и б)? Как узнать, сколько раз за неделю класс переходит из одной комнаты (кабинета) в другую (модель б)? Сколько учителей в школе (модель в)? Как определить, сколько уроков в неделю проводит данный учитель (модель в)?

27. На складе хранятся приборы 1000 наименований, не более 100 экземпляров каждого прибора. Приборы размещены в 50 помещениях, вмещающих до 2000 приборов каждое. Составьте информационную модель склада, которая позволяет узнать, есть ли на складе данный прибор и в каком помещении его искать.

28. Составьте информационные модели:

а) свидетельства о рождении;

б) паспорта;

в) аттестата о среднем образовании;

г) железнодорожного билета.

Как проверить, могут ли все перечисленные документы принадлежать одному и тому же человеку?

29 Общие величины в алгоритмическом языке

29.1. Для обращения к величинам модели они должны быть описаны

Во всех примерах предыдущего параграфа мы строили только фрагменты алгоритмов, но не полные алгоритмы с заголовками.

Дело в том, что по правилам алгоритмического языка любая величина должна быть описана прежде, чем ее можно будет использовать. Составляя фрагменты алгоритмов, мы откладывали «на потом» вопрос о том, где и как задавать величины, входящие в модель, чтобы к ним можно было обращаться в алгоритмах. Теперь пришло время в этом разобраться.

Вернемся к модели кинозала. Предположим, что мы хотим создать автоматизированную систему продажи билетов. Понятно, что для этого придется доработать модель М5, например добавить в нее возможность продажи билетов на несколько сеансов. Но главное — надо будет составить сравнительно большое количество алгоритмов, работающих с этой моделью. И в каждом из этих алгоритмов надо иметь возможность обращаться к величинам, составляющим информационную модель кинозала.

Конечно, можно обойтись уже известными нам средствами: в каждом алгоритме описать все величины модели, объявить их аргументами или результатами (в зависимости от действий, выполняемых каждым конкретным алгоритмом), и все будет работать.

Но такой подход очень неудобен. Во-первых, потребуется много раз переписывать одно и то же. Мало того, что одинаковые описания будут входить в заголовки всех алгоритмов, ведь при каждом вызове вспомогательного алгоритма величины модели придется перечислять в качестве параметров!

Во-вторых, при изменении модели (а как мы видели в предыдущем параграфе, доработка модели — процесс нормальный и неизбежный) придется вносить исправления в каждый алгоритм и в каждую команду вызова. При разработке большой системы это может оказаться очень трудоемким делом. К тому же при массовых исправлениях легко ошибиться, и значит, все ранее проверенные алгоритмы надо будет тестировать снова!

Поэтому в алгоритмический язык включены специальные средства, позволяющие не повторять многократно одни и те же описания, если несколько алгоритмов работают с одной моделью.

29.2. Общие величины

Если предполагается, что с величинами информационной модели будут работать различные алгоритмы, то они должны принадлежать одновременно всем этим алгоритмам и в то же время ни одному из них в отдельности. Такие величины называют ***общими***, или ***глобальными***.

В алгоритмическом языке можно объединить группу алгоритмов и задать для них общие величины. Конструкция, объединяющая алгоритмы и их общие величины, называется ***исполнителем***. Это название не случайно, так как данная конструкция имеет много общего с исполнителями, которых мы рассматривали ранее. Более подробно мы обсудим это в следующем параграфе.

Конструкция *исполнитель* начинается со служебного слова **исп**, после которого идет имя исполнителя. Затем следуют описания величин и алгоритмы, использующие эти величины. Заканчивается конструкция служебным словом **кон**.

Общие величины описываются *перед всеми алгоритмами*, эти описания находятся *снаружи* всех алгоритмов. Причем подчеркивается, что эти величины действительно общие, они не принадлежат ни одному алгоритму, но все алгоритмы могут к ним обращаться.

Вот пример фрагмента системы продажи билетов:

```
исп Кинозал
| ОПИСАНИЯ ОБЩИХ ВЕЛИЧИН

лог таб продано [1:28, 1:20] | продано[i, j] = да <=>          М4
                             | место j ряда i продано
цел таб цена [1:28]          | цена [i] = цена билета в ряду i
| АЛГОРИТМЫ

алг начать продажу                                            А111
  надо | все места в зале свободны
нач цел ряд, место
| нц для ряд от 1 до 28
| | нц для место от 1 до 20
| | | продано [ряд, место] :=нет
| | кц
| кц
кон
```

A 112

```
алг цел выручка
   надо | знач = выручка от проданных на сеанс билетов
нач цел i, j
   знач :=0
   нц для i от 1 до 28
      нц для j от 1 до 20
         если продано [i, j]
         | то знач :=знач+цена [j]
         все
      кц
   кц
кон

... (другие алгоритмы)
кон
```

Здесь таблицы "продано" и "цена" описаны как общие величины, поэтому их не надо описывать в каждом алгоритме. Все остальные величины, которые используются в алгоритмах, описываются как обычно. Например, для перебора всех мест в зале в алгоритме А111 описаны величины ряд и место, а в алгоритме А112 описаны величины i и j.

29.3. Размещение общих величин в памяти

Величины, помещенные в алгоритмах, размещаются в памяти, отведенной для соответствующих алгоритмов. Общие величины заносятся компьютером в ***общую память*** исполнителя, которая не принадлежит ни одному алгоритму, но доступна для всех алгоритмов исполнителя.

Память для каждого алгоритма выделяется при его вызове и освобождается, когда он заканчивает работу. Соответственно величины этого алгоритма создаются и удаляются при каждом вызове.

Память для общих величин исполнителя выделяется *один раз*, в самом начале работы системы. В дальнейшем выделяется и освобождается память алгоритмов и входящих в них величин, но общие величины остаются на месте.

Если представлять себе память компьютера в виде классной доски, то можно считать, что в самом начале она разбивается на

«большие прямоугольники» исполнителей, внутри которых также вначале отводится память для общих величин исполнителя, а потом — по ходу работы — рисуются и стираются прямоугольники алгоритмов.

При работе системы продажи билетов компьютер выделит в памяти место для таблиц "продано" и "цена", а затем начнет выполнять алгоритм "продажа билетов", по обычным правилам выделяя и освобождая память.

29.4. Начальные действия с общими величинами

В приведенном выше фрагменте мы использовали простейший вариант информационной модели кинозала, в котором количество рядов и мест заданы числами.

Если поменять эту модель на модель М5, описав ее величины как общие, мы столкнемся с проблемой при выделении памяти для таблиц: в качестве размера таблицы указана переменная, но ее значение еще неизвестно в тот момент, когда для таблицы надо выделить память.

Для выхода из этой ситуации можно задать значение величин чр и чм сразу после их описания:

```
цел чр, чм; чр :=28; чм :=20
лог таб продано [1:чр, 1:чм]
цел таб цена [1:чр]
```

Компьютер обработает эту запись следующим образом. Сначала будет выделена память для целых величин чр и чм. Затем эти величины получат значения: в ячейку чр запишется число 28, в ячейку чм число 20. После этого компьютер выделит память для таблиц "продано" и "цена". Размеры этих таблиц заданы переменными, но соответствующие величины уже имеют конкретные значения, поэтому проблем не возникнет.

В результате будут созданы точно такие же таблицы, как и в предыдущем варианте. Однако новый вариант имеет одно существенное преимущество: в алгоритмах можно указывать переменные вместо конкретных чисел, задающих количество рядов и мест. Если теперь мы захотим использовать эту систему для зала другого размера, достаточно будет изменить только первую строчку описания, а все алгоритмы останутся прежними.

В алгоритмическом языке разрешается помещать среди описания общих величин любые команды. Эти команды называются ***начальными действиями***. Они будут выполняться раньше всех алгоритмов. Выделение памяти для общих величин и выполнение начальных действий производится в том порядке, в котором они записаны.

29.5. Общие величины — особый вид величин

Вид величины — это ее информационная роль в алгоритме (п. 16.4). Мы уже знакомы с такими видами, как аргументы, результаты, промежуточные величины, значения функций.

Общие величины — еще один вид величин. От всех остальных они отличаются тем, что память для них выделяется один раз и не освобождается в процессе работы компьютера. Другое важное отличие — возможность обращения к общим величинам из различных алгоритмов.

ЗАДАЧИ И УПРАЖНЕНИЯ

1. Дополните все ранее написанные фрагменты алгоритмов работы с кинозалом заголовками и необходимыми описаниями. Включите полученные алгоритмы в исполнитель *Кинозал*.
2. На основе модели М6 и составленных для нее фрагментов алгоритмов напишите исполнителя *Диспетчер*.

30 Информационные модели исполнителей

30.1. Информационная модель обстановки на поле *Робота*

Вспомним хорошо нам знакомого исполнителя *Робот*. Чтобы полностью описать обстановку на поле *Робота*, необходимо указать размеры поля, положение самого *Робота*, распределение стен и закрашенных клеток, температуру и радиацию в каждой точке.

Формализовав все эти данные, получим информационную модель поля *Робота*:

М 20

```
цел xmax  | количество столбцов на поле
цел ymax  | количество строк на поле
цел x     | 1 <= x <= xmax, x-координата Робота (номер столбца)
цел y     | 1 <= y <= ymax, y-координата Робота (номер строки)
лог таб ниже [1: xmax, 1: ymax]   | ниже [i, j] = да <==> ниже стена
лог таб выше [1: xmax, 1: ymax]   | выше [i, j] = да <==> выше стена
лог таб левее [1: xmax, 1: ymax]  | левее [i, j] = да <==> левее стена
лог таб правее [1: xmax, 1: ymax] | правее [i, j] = да <==> правее
                                  | стена
лог таб закр [1: xmax, 1: ymax]   | закр [i, j] = да <==> клетка
                                  | закрашена
вещ таб a [1: xmax, 1: ymax]      | a[i, j] — уровень радиации
вещ таб t [1: xmax, 1: ymax]      | t[i, j] — температура в клетке
```

Пользуясь моделью М20, можно имитировать любые действия по управлению *Роботом* в прямоугольнике (табл. 24).

ТАБЛИЦА 24. **Моделирование управления *Роботом***

Управление *Роботом*	Имитация управления *Роботом*
вправо	x := x + 1
вверх	y := y – 1
закрасить	закр [x, y] := да

нц n раз
  если температура > 100
 	
	то s := s + радиация
	  все
	  вниз
	кц
		нц n раз
		  если t[x, y] > 100
		
			то s := s + a[x, y]
			  все
			  y := y + 1
			кц

если сверху стена 	то влево все	если выше [x, y] 	то x := x – 1 все
утв	Робот в левом верхнем углу	утв x = 1 и y = 1
утв	Робот в правом нижнем углу	утв x = xmax и y = ymax

30.2. Информационная модель исполнителя *Робот*

Пользуясь моделью М20, составим алгоритмы, соответствующие командам из системы команд *Робота*, и объединим их вместе с моделью в конструкцию **исп-кон** алгоритмического языка.

```
исп Робот
|  цел xmax | количество столбцов на поле
|  ...      |
|  ...  (полностью переписанная модель М20)
|
|                                              А 113
|  алг вправо
|  нач
|  |  x :=x+1
|  кон
|
|                                              А 114
|  алг лог справа стена
|  нач
|  |  знач := правее [x, y]
|  кон
|
|                                              А 115
|  алг лог справа свободно
|  нач
|  |  знач := не правее [x, y]
|  кон
|
|  ... (алгоритмы, имитирующие остальные команды Робота)
кон
```

Мы получили ***информационную модель исполнителя*** *Робот*. Она включает в себя:

а) информационную модель М20 обстановки на поле *Робота*;

б) имитации команд *Робота* в этой модели — каждая команда исполнителя оформляется в виде отдельного алгоритма.

Благодаря использованию общих величин алгоритмы "вправо", "справа стена" и др. не имеют ни аргументов, ни результатов. Их вызовы внешне неотличимы от вызовов команд *Робота*. Поэтому алгоритмы работы с такой моделью *ничем не отличаются* от алгоритмов работы с настоящим исполнителем. Используя алгоритмы этой модели, мы можем выполнить *любой* алгоритм управления *Роботом* точно так же, как сделали бы это с настоящим *Роботом*.

30.3. Исполнители в алгоритмическом языке

В общем случае информационная модель исполнителя в алгоритмическом языке записывается так:

```
исп имя
|  описание общих величин исполнителя
|  и начальных действий с ними
|  набор алгоритмов исполнителя
кон
```

Общие величины исполнителя внутри любых его алгоритмов использоваться могут, а вне исполнителя — нет.

Поскольку вызовы вспомогательных алгоритмов в алгоритмическом языке внешне неотличимы от вызовов команд исполнителей, то можно говорить не про алгоритмы исполнителя, а про его «команды», про вызовы этих команд и т. п. При составлении алгоритмов вообще не важно, существует ли исполнитель на самом деле или имитируется на компьютере. Именно поэтому конструкция алгоритмического языка носит название «исполнитель». В дальнейшем мы будем говорить «исполнитель», не уточняя, идет ли речь о настоящем исполнителе или о его информационной модели.

30.4. Моделирование отказов

Как мы знаем, в некоторых ситуациях у исполнителя может возникнуть отказ, например, при попытке выполнить какую-то команду в неподходящих для нее условиях. Эту особенность можно учесть и при построении модели: в алгоритм, реализующий команду исполнителя, вставляется проверка соответствующего условия (обычно с помощью **дано** или **утв**).

Например, алгоритм А114 никак не учитывает тот факт, что *Робот* не может ходить сквозь стены, а при попытке сделать это возникает отказ. Исправим алгоритм, добавив в него проверку:

```
алг вправо                                   А 116
   дано справа свободно
нач
|  x :=x+1
кон
```

Теперь если справа от *Робота* находится стена, то при попытке сделать шаг вправо возникнет отказ, как это и должно быть.

30.5. Использование исполнителей при составлении алгоритмов

Рассмотрим исполнителя — назовем его *И1* — который умеет выполнять три команды:

```
        запомнить (арг вещ x)
цел число больших (арг вещ a0)
вещ минимум
```

При выполнении команды запомнить исполнитель *И1* сохраняет аргумент команды в своей памяти. По команде

число больших(a0)

исполнитель *И1* просматривает все запомненные числа и определяет, сколько среди них чисел бóльших, чем *a*0. По команде минимум *И1* сообщает минимальное из запомненных чисел.

Исполнитель *И1* можно использовать при решении самых разных задач: например, чтобы подсчитать число клеток коридора, уровень радиации в которых выше, чем в клетке выхода:

A 117

```
алг число опасных клеток (рез цел n)
    дано | Робот в левой клетке горизонтального коридора
    надо | Робот вышел из коридора вправо, n = число клеток,
         | уровень радиации в которых выше, чем в клетке выхода
нач
|   нц пока снизу стена
|   | запомнить (радиация); вправо
|   кц
|   n := число больших (радиация)
кон
```

30.6. Задание исполнителя *И1* на алгоритмическом языке

Исполнитель *И1* пока существует только в нашем воображении, ведь мы его придумали. Если мы действительно хотим поручить компьютеру выполнение алгоритма А117, то надо либо изготовить реального исполнителя *И1* и подключить его к компьютеру, либо — что гораздо проще — подменить исполнителя *И1* его информационной моделью (т. е. заставить компьютер выполнять еще и команды исполнителя *И1*). Второй подход, вместо изготовления *И1* «в металле», требует лишь информационного ***описания*** *И1* на алгоритмическом языке:

```
исп И1
|   цел N; N := 100  | максимальное количество чисел
|   вещ таб a[1:N]
|   цел n            | количество запомненных чисел
|   n := 0
|
|   алг запомнить (арг вещ x)                                  A 118
|     дано n < N
|     надо | к запомненным числам добавлено число x
|   нач
|   | n := n + 1; a[n] := x
|   кон
|
|   алг цел число больших (арг вещ a0)                         A 119
|     надо | знач = количество чисел больших, чем a0
|   нач цел i
|   |   знач :=0
|   |   нц для i от 1 до n
|   |   | если a[i] >a0 то знач := знач +1 все
|   |   кц
|   кон
|
|   алг вещ минимум                                            A 120
|     дано n > 0
|     надо | знач = минимальное из запомненных чисел
|   нач цел i
|   |   знач := a[1]
|   |   нц для i от 2 до n
|   |   | если a[i] < знач то знач := a[i] все
|   |   кц
|   кон
кон
```

30.7. Использование исполнителей при решении чисто информационных задач

Информационные модели исполнителей могут использоваться и при решении чисто информационных задач, когда в условии задачи никакие исполнители вообще не фигурируют. Пусть, на-

пример, в прямоугольной таблице чисел **вещ таб** t[1:60, 1:60] требуется в каждом столбце найти максимальное число, а среди этих 60 максимальных чисел — минимальное. С помощью *И1* это можно сделать так:

A 121

```
алг вещ минимакс (арг вещ таб t[1: 60, 1: 60])
   надо | в каждом столбце найти максимальное число
        | и установить знач равным минимальному из них
нач цел j
|  нц для j от 1 до 60
|  |  запомнить (макс (t, j))
|  кон
|  знач := минимум
кон
```

Здесь "запомнить" и "минимум" — вызовы команд исполнителя *И1*, а "макс (t, j)" — вызов вспомогательного алгоритма-функции, который должен вычислять максимальное значение в *j*-м столбце таблицы *t*. Этот алгоритм-функцию можно записать так:

A 122

```
алг вещ макс (арг вещ таб t[1: 60, 1: 60], цел j)
   надо | знач = максимальное значение в j-м столбце таблицы t
нач цел i
|  знач := t[1, j]
|  нц для i от 2 до 60
|  |  знач := max (знач, t[i, j])
|  кц
кон
```

30.8. Информационная модель исполнителя *Стековый Калькулятор*

Если нужно реально изменить что-то в окружающем мире (перевести спутник с одной орбиты на другую или *Робота* из одной клетки в другую), то тут без внешнего, настоящего исполнителя не обойтись. Однако, когда требуется просто что-то подсчитать, всегда можно составить информационную модель соответствующего исполнителя и с ней работать.

Вспомните, например, исполнителя *Стековый Калькулятор* из § 7. Если нам понадобится такого рода исполнитель для каких-то расчетов, мы можем реализовать его на алгоритмическом языке (предполагая, что в стеке может быть не более 100 чисел):


```
исп Стековый Калькулятор
|  цел nmax; nmax :=100 | максимальное количество чисел в стеке
|  цел n; n :=0         | фактическое количество чисел в стеке
|  цел таб s[1: nmax]   | стек: в s[1] — первое число; в s[2] —
|                       | второе; ..., в s[n] — последнее
|
|  алг запомнить 1
|    дано n < nmax | (в стеке есть свободное место)
|    надо  | число 1 добавлено в стек
|  нач
|  |  n :=n+1
|  |  s[n] :=1
|  кон
|   ...
|
|  алг сложить
|    дано n > 1 | (в стеке не меньше двух чисел)
|    надо  | вместо двух верхних элементов стека помещена их сумма
|  нач цел r
|  |  r := s[n-1]+s[n] |  результат сложения
|  |  n := n-1         | уменьшение числа элементов в стеке на 1
|  |  s[n] :=r         | замена верхнего элемента стека на результат
|  кон
|
|  алг вычесть
|    дано n > 1 | (в стеке не меньше двух чисел)
|    надо  | вместо двух верхних элементов стека помещена их разность
|  нач цел r
|  |  r := s[n-1]-s[n] |  результат вычитания
|  |  n := n-1         | уменьшение числа элементов в стеке на 1
|  |  s[n] :=r         | замена верхнего элемента стека на результат
|  кон
|   ...
|
|  алг показать
|    дано n > 0 | (в стеке есть хотя бы одно число)
|    надо  | на экран выведено значение s[n]
|  нач
|  |  вывод s[n]
|  кон
кон
```

Способ запоминания чисел в приведенной выше модели называется *реализацией стека на базе линейной памяти* (*таблицы*).

30.9. Создание одних исполнителей на базе других

Создавая информационную модель исполнителя, можно использовать другого исполнителя, реального или тоже смоделированного. Такой прием называется *созданием одного исполнителя на базе другого*.

Рассмотрим пример. При обучении информатике младших школьников часто используют язык Лого, в котором есть исполнитель *Черепашка*. *Черепашка* может двигаться вперед и назад на заданное расстояние, поворачиваться направо и налево на заданное число градусов, оставлять или не оставлять след при движении.

Вот система команд этого исполнителя, записанная по правилам алгоритмического языка:

```
вперед (арг вещ x)
назад (арг вещ x)
направо (арг вещ a)
налево (арг вещ a)
со следом
без следа
```

В книгах по Лого приводится много красивых изображений, которые получены с помощью этого исполнителя. Как нам построить эти изображения, если у нас нет *Черепашки*, но есть *Чертежник*?

Можно попробовать построить информационную модель *Черепашки* на базе *Чертежника*. След смоделируем с помощью поднятия и опускания пера, движению *Черепашки* будет соответствовать движение пера *Чертежника*. У *Черепашки* есть только команды относительного смещения, значит, будем использовать команду сместиться на вектор. Из рисунка 107 видно, что координаты в команде *Чертежника* и смещение *Черепашки* связаны простыми тригонометрическими соотношениями. Угол, который фигурирует в этих формулах, — это текущий угол поворота *Черепашки*. Его можно запомнить как общую величину исполнителя и изменять по командам направо и налево. При этом необходимо учесть, что при

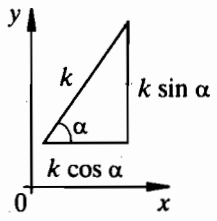


Рис. 107

работе с *Черепашкой* принято измерять углы в градусах, а для обращения к функциям алгоритмического языка их надо задавать в радианах.

Теперь мы можем записать информационную модель *Черепашки* на базе *Чертежника*.

```
исп Черепашка
| вещ угол | текущий угол движения Черепашки в градусах
| угол :=90 | сначала Черепашка смотрит вверх
|
| алг вперед (арг вещ k)                                    A 123
| нач вещ угол1 | угол в радианах
| | угол1 :=угол*3.14/180
| | сместиться на вектор (k*cos (угол1), k*sin (угол1))
| кон
|
| алг назад (арг вещ k)                                     A 124
| нач
| | вперед (–k)
| кон
|
| алг направо (арг вещ a)                                   A 125
| нач
| | угол :=угол–a
| кон
|
| алг налево (арг вещ a)                                    A 126
| нач
| | угол :=угол+a
| кон
|
| алг со следом                                             A 127
| нач
| | опустить перо
| кон
|
| алг без следа                                             A 128
| нач
| | поднять перо
| кон
кон
```

30.10. Метод последовательного уточнения с использованием исполнителей

Метод последовательного уточнения применим и в случае применения вспомогательных исполнителей. Иными словами: ***при составлении алгоритма можно использовать вызовы команд еще не существующих исполнителей, а потом описать информационные модели этих исполнителей на алгоритмическом языке***.

Разумеется, при составлении алгоритмов такого ***вспомогательного*** исполнителя можно придумать новых (более простых) исполнителей и т. д. В самом конце этой цепочки могут быть (а могут и не быть) реальные исполнители (*Робот*, ядерный реактор, самолет и пр.). При выполнении компьютер просто будет вызывать из одних алгоритмов другие, имитируя всех придуманных нами исполнителей в соответствии с их описаниями на алгоритмическом языке.

Такой метод составления алгоритмов — ***метод последовательного уточнения с использованием исполнителей*** — в настоящее время является ***основным методом алгоритмизации сложных задач***.

ЗАДАЧИ И УПРАЖНЕНИЯ

1. Информация о расположении стен, хранящаяся в таблицах «выше», «ниже», «правее» и «левее» модели М20, избыточна. Например, если правее [1, 1] = **да** (т. е. между 1-й и 2-й клетками первого ряда стена), то левее [2, 1] тоже обязательно равно **да**. Измените модель так, чтобы информация о стенах кодировалась более экономно, без избыточности.

2. Доделайте информационную модель *Робота*, добавив реализацию алгоритмами всех еще не реализованных команд этого исполнителя.

3. Добавьте в исполнитель *И1* команды:

   а) забыть все — при ее выполнении исполнитель будет стирать все запомненные числа из своей памяти;

   б) **лог** среди запомненных есть (**арг вещ** а0) — должна отвечать **да**, если среди запомненных чисел есть число а0, и **нет** в противном случае;

   в) **цел** число равных (**арг вещ** а0) — выдает количество чисел среди запомненных, равных а0;

   г) стереть (**арг вещ** х) — при ее выполнении из памяти стирается любое из запомненных чисел, равных *х* (если такого числа в памяти *И1* нет, то память останется прежней, но ни отказа, ни зацикливания возникать не должно).

4. Измените исполнителя *И1* так, чтобы получился исполнитель *И2* со следующей системой команд:

```
   запомнить (арг вещ х)| число добавляется к запомненным
цел число запомненных
   забыть все
вещ последнее           | последнее запомненное число
   сложить два последних
   умножить два последних
```

   При выполнении команд сложения и умножения *И2* должен два последних запомненных числа заменить их суммой или произведением (количество запомненных чисел при этом уменьшится на 1).

5. Известно, что исполнитель *И2* (упр. 4) запомнил коэффициенты квадратного трехчлена, начиная со свободного члена. Составьте алгоритм, вычисляющий значение этого трехчлена в точке *х* (состояние *И2* после выполнения алгоритма может быть любым).

6. Известно, что исполнитель *И2* (упр. 4) запомнил коэффициенты некоторого многочлена, начиная со свободного члена. Составьте алгоритм, вычисляющий значение этого многочлена в точке *х* (состояние *И2* после выполнения алгоритма может быть любым).

   Указание. Используйте схему Горнера вычисления значения многочлена в точке.

7. Известно, что исполнитель *И2* (упр. 4) запомнил 10 чисел. Составьте алгоритм, который найдет первое запомненное число (состояние *И2* после выполнения алгоритма может быть любым).

8. Опишите, как будут меняться общие величины исполнителя *Черепашка* в процессе выполнения алгоритма

A 129

```
алг след
    надо | на песке нарисован квадрат со стороной 1
нач
| со следом
| вперед (1); направо (90)
| вперед (1); направо (90)
| вперед (1); направо (90)
| вперед (1); направо (90)
| без следа
кон
```

9. Нарисуйте след, который оставит *Черепашка* после выполнения команд:

а)
```
нц для к от 1 до 10
| вперед (к)
| направо (90)
кц
```

б)
```
нц для к от 1 до 10
| вперед (1)
| направо (90)
| вперед (к)
| направо (90)
кц
```

в)
```
нц для к от 1 до 10
| вперед (к)
| направо (90)
| вперед (к + 1)
| направо (90)
кц
```

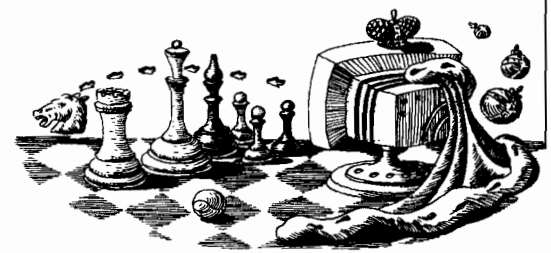
10. Составьте алгоритм для *Черепашки*, после выполнения которого будут нарисованы:
а) спираль из *n* звеньев;
б) правильный шестиугольник;
в) правильный пятиугольник;
г) пятиугольная звезда;
д) правильный *n*-угольник со стороной 1.

11. Постройте полную информационную модель исполнителя *Стековый Калькулятор*.

12. Постройте полную информационную модель исполнителя *Счетчик*.

ГЛАВА 6

Как устроены компьютерные программы



Сегодня компьютеры окружают человека повсюду. Мы уже говорили об их главных применениях в главе 1, возможно, вы и сами встречаетесь с компьютерами в повседневной жизни. В этой главе мы расскажем об основных принципах, лежащих в основе реальных компьютерных программ. Разумеется, нет никакой возможности сколько-нибудь подробно описать в учебнике реальные применения ЭВМ и соответствующие программы. Мы проиллюстрируем отдельные методы обработки информации в различных областях на простейших примерах (т. е., рассказывая об океанских лайнерах, продемонстрируем корыто и объясним, почему оно плавает).

31 Информационные системы

С общими принципами работы информационных систем мы уже знакомились в § 4. Методы хранения и обработки информации в реальных информационных системах достаточно сложны. Трудозатраты на создание таких систем измеряются десятками и даже сотнями человеколет. Вместе с тем общие идеи этих систем можно понять на простых, «игрушечных» примерах.

31.1. Учебная информационная система *Вагон*

Рассмотрим информационную систему *Вагон* для хранения информации о свободных и занятых местах в одном 36-местном купированном вагоне, следующем без остановок по маршруту Москва — Петушки. Как создать такую систему? Прежде всего построим информационную модель вагона:

```
исп Вагон
| цел N; N :=36      | количество мест в вагоне
| цел таб M[1:N]     | M[i] = 0 <==> место i свободно
|                    | M[i] = 1 <==> место i продано
```

Теперь покажем, как могут выглядеть алгоритмы, использующие общую величину *M* и выполняющие обработку информации в системе *Вагон*:

A 130

```
| алг учесть продажу места (арг цел i)
|   дано 1 <=i <=N и M[i] = 0
| нач
| | M[i] :=1
| кон
```

A 131

```
| алг учесть возврат билета на место (арг цел i)
|   дано 1 <=i <=N и M[i] = 1
| нач
| | M[i] :=0
| кон
```

A 132

```
| алг цел количество проданных мест
| нач цел i
| | знач :=0
| | нц для i от 1 до N
| | | знач := знач + M[i]
| | кц
| кон
```

A 133

```
| алг цел номер свободного места
|   дано | свободные места есть
|   надо M[знач] = 0
| нач
| | знач := 1
| | нц пока M[знач] <> 0
| | | знач := знач + 1
| | кц
| кон
|  ... ... ...
кон
```


Мы рассмотрели работу с одним вагоном. Если требуется закодировать поезд, состоящий из *K* таких вагонов, то можно, например, использовать прямоугольную таблицу

```
цел таб M[1:K, 1:N]
```

где первый индекс означает номер вагона, а второй — номер места в вагоне.

31.2. Учебная информационная система *Телефонная книжка*

Рассмотрим еще один пример «игрушечной» информационной системы. Пусть требуется создать справочную систему обо всех владельцах телефонов в маленьком городе (число телефонов меньше 1000). Эту информацию можно закодировать так:

```
исп Телефонная книжка
| цел n; n :=0          | число телефонов в городе
| лит таб ФИО[1:999]    | ФИО владельца телефона с номером i.
|                       | если телефона i нет, то ФИО[i]= " "
```

Приведем несколько алгоритмов из этого исполнителя:

A 134

```
| алг учесть установку нового телефона (арг цел t, лит f)
|   дано ФИО[t] = " "| t — номер телефона, f — ФИО абонента
|   надо |информация о новом телефоне запомнена
| нач
| | n := n + 1
| | ФИО[t] := f
| кон
```

A 135

```
| алг смена номера (арг цел tстар, tнов)
|   дано ФИО [tнов] = " "|tстар — старый номер, tнов — новый
|   надо | соответствующая информация изменена
| нач цел i
| | ФИО[tнов] := ФИО[tстар]
| | ФИО[tстар] := " "
| кон
```

А вот как может выглядеть алгоритм, который запрашивает у человека ФИО абонента и выдает в ответ номер его телефона:

A 136

```
алг справка о номере телефона
нач лит f, цел t
  вывод "Укажите ФИО абонента : "; ввод f
  поиск абонента (f, t)
  если t = 0
    то вывод "Абонент не найден"
    иначе вывод "Абонент", f, "имеет телефон", t
  все
кон
```

Здесь использован вспомогательный алгоритм "поиск абонента", который по значению элемента *f* должен определить его индекс *t* в таблице ФИО. Этот алгоритм аналогичен А88:

A 137

```
алг поиск абонента (арг лит f, рез цел t)
 надо | ФИО[t] = f, если значение f встречается в таблице ФИО,
      | t = 0 в противном случае
нач цел i
  t :=0
  нц для i от 1 до 999
  | если ФИО[i]=f то t :=i все
  кц
кон
  ...
кон
```

ЗАДАЧИ И УПРАЖНЕНИЯ

1. Добавьте в исполнитель *Вагон* следующие команды:
   а) начать работу (все места должны стать свободными);
   б) **лог** есть место на нижней полке;
   в) найти два свободных места в одном купе (**рез цел** n1, n2).

2. Пусть информация о проданных билетах и свободных местах в поезде хранится в таблице

   лог таб свободно [1:K, 1:N]

   (свободно [i, j] = **да**, если *j*-е место в *i*-м вагоне свободно).

   Выполните упражнение 1 применительно к поезду (в алгоритме 1, в) добавьте еще один результат — номер вагона).

3. Придумайте информационную модель поезда, в котором первые 8 вагонов — купейные, 9-й — вагон-ресторан, а вагоны с 10-го по 14-й — плацкартные. Пусть ваш класс в полном составе собрался поехать на экскурсию. Сформулируйте требования, которым, по вашему мнению, должны удовлетворять места в поезде для такой поездки (например, «все места в одном вагоне», «не более чем в двух соседних вагонах», «рядом с рестораном» и т. п.). Составьте алгоритм, определяющий, есть ли в поезде нужное количество мест, отвечающих вашим требованиям.

4. Перепишите алгоритм А137, взяв за основу вместо алгоритма А88 алгоритм А89.

5. Пусть в системе *Телефонная книжка* информация о владельцах телефонов задается следующими общими величинами:

цел n	общее количество телефонов
лит таб ФИО [1:999]	ФИО владельца i-го телефона
цел таб t[1:999]	номер i-го телефона

   а) измените алгоритмы А135—А137 для этой информационной модели;
   б) пусть абоненты в таблице ФИО уже упорядочены по алфавиту. Составьте алгоритм установки нового телефона так, чтобы после его выполнения свойство упорядоченности таблицы ФИО сохранялось.

6. В рамках исполнителя *Телефонная книжка* придумайте способ кодирования информации об адресах абонентов. Составьте алгоритм, который по адресу дома определяет количество телефонов, установленных в доме.

32 Обработка текстов

В этом параграфе мы рассмотрим простейший пример устройства текстового редактора.

32.1. Текст, курсор и окно

Обычно при работе с системами обработки текстов экран служит как бы окном, через которое человек глядит на текст. Текст при этом удобно представлять в виде длинного бумажного свитка, расположенного «за окном» (рис. 108, *а*).

Курсор показывает место в тексте, в котором можно вставлять, удалять или изменять символы.

Курсор можно двигать по тексту, нажимая на клавиши со стрелками. При перемещении за пределы окна курсор «упирает-

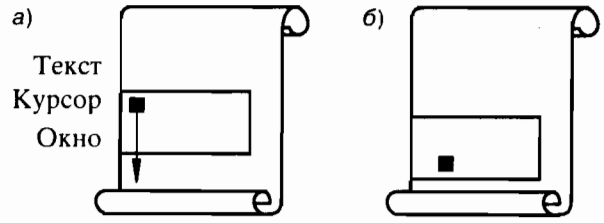


Рис. 108

ся» в край окна и сдвигает его (рис. 108, *б*). Конечно, экран монитора при этом остается на месте — просто меняется изображаемый текст. При движении курсора вниз текст смещается по экрану вверх, а при движении курсора вверх текст смещается вниз.

32.2. Учебная модель редактора текстов

Рассмотрим простейший случай. Пусть текст целиком помещается в оперативной памяти компьютера и состоит не более чем из 200 строк размером не более 100 символов каждая, а окно вмещает 24 строки по 80 символов. Построим информационную модель текста, окна и курсора (рис. 109):

```
исп Простейший редактор
| цел L, C; L = 200; C = 100; | максимальные размеры текста:
|                             | не больше L строк по C символов
| цел Xmax, Ymax; | размер экрана:
| Xmax := 80; Ymax := 24 | Ymax строк по Xmax символов
| цел N | число строк в тексте
| сим таб T[1:L; 1:C] | i-я строка хранится в T[i, 1:C]
|
| цел Yo, Xo | положение окна:
|            | Yo — число строк текста, расположенных выше окна
|            | Xo — число позиций (символов) текста левее окна
| цел Yk, Xk | координаты курсора в окне:
|            | 1 <= Yk <= Ymax — номер строки (сверху вниз),
|            | 1 <= Xk <= Xmax — номер позиции в строке
```

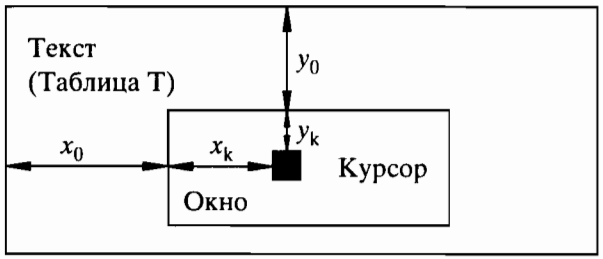


Рис. 109

Зная значения величин Yo и Yk, легко определить номер строки текста, на которой стоит курсор, — это y = Yo + Yk. Аналогично, позиция курсора в тексте — x = Xo + Xk. Также легко узнать, какие строки видны на экране: первая строка имеет номер Yo + 1, последняя Yo + Ymax (конечно, если такая строка есть, т. е. если Yo + Ymax <= N). В каждой строке изображается Xmax символов: от Xo + 1 до Xo + Xmax. Приведем пару алгоритмов:

А 138

```
| алг вниз на строку
|   надо | курсор смещен вниз на строку, если нужно, сдвинуто
|        | и окно. Если курсор уже был за последней строкой
|        | текста и сдвинуть его нельзя, то звонок
| нач
|   выбор
|     при Yo+Yk>N : звонок  | курсор ниже текста => звонок
|     при Yk<Ymax : Yk :=Yk+1  | сдвиг курсора внутри окна
|     иначе Yo :=Yo+1  | курсор уперся => сдвиг окна
|   все
| кон
```

Стандартная команда алгоритмического языка звонок, которую мы здесь использовали, приводит к выдаче звукового сигнала.

А 139

```
| алг удалить символ
|   надо | удален символ текста, на котором стоял курсор.
|        | Символы правее курсора сдвинуты на одну позицию
|        | влево, справа добавлен пробел
| нач цел y, i
|   y :=Yo+Yk  | номер строки в таблице T
|   нц для i от Xo + Xk до C -1 | сдвиг символов правее курсора
|   |  T[y, i] :=T[y, i+1]      | на одну позицию влево
|   кц
|   T[y, C] := ' '| справа добавляется один пробел
| кон
```

В алгоритмах А138 и А139 меняются только значения общих величин информационной модели. При работе с редактором текстов эти изменения надо еще отобразить на экране:

А 140

```
алг переизобразить экран и курсор
нач цел m, i
| очистить экран
| m :=min (Ymax, N–Y0) | число строк текста, попавших в окно
| нц для i от 1 до m
|   поз(1, 1); изобразить строку (Yo+i)
| кц
| поз(Yk, Xk) | изображение курсора в нужной позиции
кон
```

Здесь используются вспомогательный алгоритм "очистить экран" (см. § 26, упр. 3), команда исполнителя *Экран* поз (см. п. 26.7) и вспомогательный алгоритм "изобразить строку":

А 141

```
алг изобразить строку (арг цел y)
  дано | y — номер строки в таблице T
  надо | на экране изображена попадающая в окно часть строки,
       | т. е. символы от Xo + 1 до Xo + Xmax
нач цел j
| нц для j от Xo+1 до Xo + Xmax
|   вывсим (T[y, j]) | команда вывсим описана в п. 26.7
| кц
кон

... (алгоритмы других команд редактора)

кон
```

Итак, мы разобрали информационную модель текста и работу алгоритмов, которые его меняют, изображая изменения на экране. Кроме этой основной части, записанной на алгоритмическом языке с помощью конструкции *исполнитель*, текстовый редактор включает в себя основной алгоритм (его мы не приводим), при выполнении которого компьютер ждет, пока человек нажмет на клавишу (команда клав п. 26.8); затем вызывает (выполняет) соответствующий этой клавише алгоритм (например, А138, если человек нажал на стрелку вниз); изображает результат на экране (т. е. вызывает алгоритм А140); снова ждет,

пока человек нажмет клавишу, и т. д. Этот процесс прекращается по специальной команде окончания редактирования. Кроме того, основной алгоритм читает текст с диска перед началом и записывает измененный текст на диск в конце работы.

ЗАДАЧИ И УПРАЖНЕНИЯ

1. В рамках исполнителя *Простейший редактор* составьте алгоритмы со следующими заголовками:

а) **алг** в начало строки
   надо | курсор в начале строки

б) **алг** удалить конец строки
   надо | удалены все символы строки правее курсора
   | (т. е. заменены на пробелы)

в) **алг** в начало строки по непробелам
   надо | курсор на месте первого непробела в строке

г) **алг** в конец строки по непробелам
   надо | курсор за последним непробелом в строке

д) **алг** влево на позицию
   надо | курсор смещен на одну позицию влево (если он
   | уже был в первой позиции, то звонок)

е) **алг** вверх на строку
   надо | курсор смещен на одну строку вверх (если он уже
   | был в первой строке, то звонок)

ж) **алг** в начало следующей строки
   надо | курсор смещен вниз в начало следующей строки
   | (если надо, сдвинуто и окно). Если курсор уже
   | был за последней строкой текста, то звонок

з) **алг** центрирование строки
   надо | текст в строке расположен примерно по центру,
   | т. е. число пробелов слева и справа от текста от-
   | личается не более чем на 1

и) **алг** на слово вправо
   надо | курсор смещен вправо в начало следующего
   | слова, т. е. находится под первым символом слова

2. Придумайте другие команды текстового редактора. Составьте соответствующие этим командам алгоритмы в рамках исполнителя *Простейший редактор*.

3. Известно, что ширина текста не превышает 63 символов в строке. Как упростить исполнителя *Простейший редактор* для этого случая?

4. Придумайте какую-нибудь другую информационную модель текста, окна и курсора. Опишите преимущества и недостатки вашей модели по сравнению с моделью, приведенной в учебнике.

33 Научно-технические расчеты

33.1. Компьютер — вычислительная машина

Один из древнейших видов работы с информацией — вычисления. Земледельцам, строителям, мореплавателям, торговцам приходилось проводить те или иные подсчеты для определения, например, площади поля, размеров строительных блоков, местонахождения корабля и т. п. По мере развития науки и техники, по мере усложнения машин и сооружений потребность в вычислениях возрастала, они становились все более громоздкими. Достаточно сказать, что расчет запуска ракеты на нужную орбиту требует выполнения нескольких миллионов арифметических действий.

Именно потребности в автоматизации вычислений первоначально привели к появлению ЭВМ — электронных ***вычислительных*** машин. Современное название ***компьютер*** также происходит от английского *compute* — считать, вычислять. Сначала компьютер служил для вычислений и научно-технических расчетов, и лишь позже его начали применять как универсальную машину для обработки информации.

Вычислительные возможности компьютера используются во многих областях: при расчетах зарплаты и прочности автомобиля, при управлении самолетом или ракетой и при расшифровке древних надписей и т. п. Применение уникальных вычислительных возможностей компьютера позволило создать ряд принципиально новых приборов и устройств, без компьютера вообще немыслимых. В этом параграфе мы расскажем только про одно из таких устройств — томограф.

33.2. Томография

Все вы знаете о рентгене — методе диагностики заболеваний с помощью просвечивания человека рентгеновскими лучами. На рентгеновском снимке врач видит «тень», отброшенную органами человека, и может безошибочно распознать трещины и переломы костей, вывихи, наличие посторонних предметов и т. д. Однако обнаружить, например, опухоль мозга с помощью рентгена практически невозможно, так как черепные кости экранируют внутренние ткани мозга. Поэтому вплоть до недавнего времени при подозрении на опухоль мозга врач вынужден был прибегать к вскрытию черепа пациента — иногда лишь для того, чтобы убедиться, что подозрения были напрасными!

Томограф (от греческого корня *tomo* — срез) позволяет врачу получить не «тень» от просвечивания, а изображение «среза» человеческого тела. Как устроен томограф? Обычно он представляет собой кольцо диаметром около метра, на котором укреплено несколько сотен источников рентгеновских лучей и несколько сотен датчиков. Внутрь кольца помещается человек, после чего последовательно один за другим на десятые доли секунды включаются источники. При каждом включении очередного источника со всех датчиков снимаются показания интенсивности рентгеновских лучей, прошедших через человеческий организм в разных направлениях (от источника к датчикам). После включения всех источников получаются сотни тысяч чисел. Эти числа сложным образом обрабатываются на компьютере, и компьютер изображает срез (плотность) человеческого тела (как если бы вместо измерений кольцо просто «разрезало» человека).

Обработка результатов измерений на компьютере требует выполнения нескольких миллиардов операций сложения и умножения многозначных чисел. Поэтому своим появлением на свет томография обязана прежде всего компьютерам. Современные специализированные компьютеры в состоянии изготовить томограмму за время порядка секунды.

Получив несколько соседних «срезов», можно с помощью компьютера синтезировать и трехмерное (объемное) изображение внутренних органов человека, опухолей и т. п. Томограф помогает не только диагностировать, но и лечить, например ввести лекарство точно в нужную точку и тем самым обойтись без хирургической операции.

Наконец, сфера применения томографа не ограничивается только медициной. Вместо человека в томограф можно поместить техническое изделие и «диагностировать» в нем внутренние, скрытые дефекты, трещины и т. п.

33.3. Приближенные вычисления

Вычисления на компьютере обладают некоторой спецификой, отличающей их от привычных вам вычислений в математике. Рассмотрим, например, задачу вычисления корня уравнения $f(x) = 0$. В курсе школьной математики вы рассматривали только такие уравнения, для которых ответ можно вычислить по формуле, так, для уравнения $ax^2 + bx + c = 0$ известна формула:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Во многих случаях, однако, ответ не выражается формулой. Например, для корня уравнения $\cos(x) = x$ (т. е. для корня функции $f(x) = \cos(x) - x$) подобной простой формулы нет. В этом случае можно пытаться, *не выводя точных формул*, вычислить корень ***приближенно***, например с точностью до 0,001. Несколько методов такого приближенного вычисления разных математических величин мы сейчас рассмотрим, и в частности один из методов определения корня. Подчеркнем, что мы найдем корень, не имея для этого точных формул и не выводя их. Наш ответ будет неточным (приближенным), но зато мы сможем один и тот же метод применить к самым разным уравнениям, в том числе и к таким, которые математика точно решать не умеет.

33.4. Вычисление корня функции методом деления отрезка пополам

Пусть f — непрерывная функция и $f(a) * f(b) <= 0$. Тогда f обязательно имеет корень на отрезке $[a, b]$. Возьмем середину отрезка $c = (a + b)/2$ в качестве приближенного значения корня. Настоящий корень отличается от c не более чем на половину длины отрезка, т. е. не более чем на $(b - a)/2$. Если такая точность нас не устраивает, то можно от отрезка $[a, b]$ перейти к одной из его половин: либо к $[a, c]$, либо к $[c, b]$, а именно:

если $f(a) * f(c) <= 0$, то корень на отрезке $[a, c]$;

если $f(c) * f(b) <= 0$, то корень на отрезке $[c, b]$.

Так как длина нового отрезка вдвое меньше старого, то, взяв в качестве приближенного значения корня середину нового отрезка, мы получим корень с точностью $(b - a)/4$. Если и эта точность нас не устраивает, поделим пополам новый отрезок и т. д.

Таким образом, мы можем делить отрезок пополам и переходить к одной из его половин, пока длина отрезка не станет достаточно малой, а потом в качестве корня взять середину отрезка. Соответствующий алгоритм приближенного вычисления корня записывается так:

A 142

```
алг вещ корень (арг вещ A, B, d)
   дано A < B и d > 0 и F(A)*F(B) <=0
   надо | знач ≈ корень F(x) =0 на отрезке [A, B] с точностью d
нач вещ a, b, c
| a :=A; b :=B
| нц пока (b-a)/2 > d | пока требуемая точность не достигнута
| | c := (a+b)/2      | c := середина отрезка [a, b]
| | если F(a) * F(c) <=0
| | | то b :=c        | переход к отрезку [a, c]
| | | иначе a :=c     | переход к отрезку [c, b]
| | все
| кц
| утв F(a)*F(b) <=0 и (b-a)/2 <=d
| знач := (a+b)/2
кон
```

Чтобы выполнить этот алгоритм, надо задать вспомогательный алгоритм-функцию с именем F. Для уравнения $\cos(x) = x$, например, можно написать:

A 143

```
алг вещ F (арг вещ x)
нач
| знач := cos (x) - x
кон
```

После этого для вычисления корня уравнения $\cos(x) = x$ на отрезке [0, 2] с точностью до 0,001 достаточно написать x := корень (0, 2, 0.001).

33.5. Приближенное вычисление площади методом трапеций

В математике часто возникает необходимость вычислить площадь под графиком какой-то функции. Эта площадь называется ***интегралом функции***. С понятием интеграла вы познакомитесь

на уроках математики в 11 классе, но в приближенном вычислении интегралов с помощью компьютера сможете разобраться уже сейчас.

Для приближенного вычисления интеграла от функции на отрезке (т. е. для вычисления площади под графиком функции; рис. 110) разобьем отрезок $[a, b]$ на n равных частей (например, на $n = 10$) и заменим график $f(x)$ ломаной, состоящей из n звеньев (подобно тому, как мы поступали при рисовании параболы в § 16).

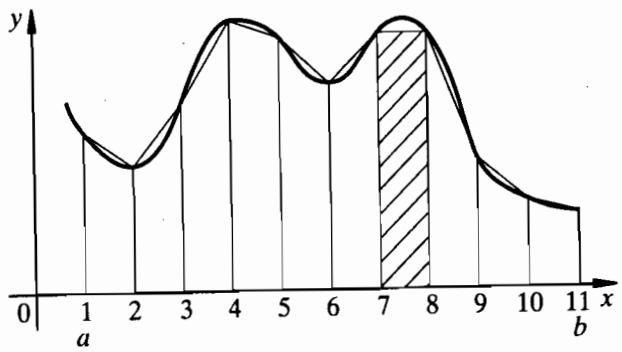


Рис. 110

Тогда искомую площадь можно приближенно вычислить как площадь под ломаной, т. е. как сумму площадей получившихся маленьких трапеций:

$$S \approx S_1 + S_2 + S_3 + \ldots + S_n,$$

где площадь одной трапеции, например S_7, можно вычислить, умножив полусумму оснований $f(x_7)$ и $f(x_8)$ на высоту $h = (b-a)/n$:

$$S_7 = \frac{f(x_7) + f(x_8)}{2} * h.$$

Подставим выражения для $S_1, S_2, \ldots, S_n$ в формулу для S:

$$S \approx \left(\frac{f(x_1)}{2} + f(x_2) + f(x_3) + \ldots + f(x_n) + \frac{f(x_{n+1})}{2} \right) * h.$$

Алгоритм вычисления площади по этой формуле можно записать так:


```
алг вещ площадь (арг вещ a, b, цел n)
нач вещ s, h, цел i
| h := (b–a)/n
| s := f(a)/2 + f(b)/2
| x := a
| нц n – 1 раз
| | x :=x+h | переход к следующей точке
| | s :=s+f(x)
| кц
| знач :=s*h
кон
```

С увеличением n «зазор» между ломаной и графиком функции $f(x)$ уменьшается и вычисленная с помощью алгоритма "площадь" величина становится все ближе к значению интеграла функции на отрезке.

33.6. Метод Монте-Карло

Рассмотрим еще один интересный метод приближенного вычисления площади — ***метод Монте-Карло***. Пусть у нас есть какая-нибудь фигура на плоскости, расположенная внутри квадрата со сторонами, параллельными координатным осям (рис. 111). И пусть про любую точку квадрата мы можем быстро узнать, попадает она внутрь фигуры или нет. Тогда площадь можно вычислять так: засыплем квадрат ровным слоем песка. Ясно, что число песчинок, которые окажутся внутри фигуры, пропорционально ее площади: больше площадь — больше песчинок; меньше площадь — меньше песчинок. Поэтому, поделив количество песчинок, находящихся внутри фигуры, на количество всех песчинок в квадрате, мы можем найти, какую часть площади квадрата занимает фигура. Ну а площадь квадрата вычислить легко.

Как записать эту идею на алгоритмическом языке? Что такое «песчинки» и как засыпать квадрат ровным слоем? Для этого используются так называемые ***случайные числа***. В алгоритмическом языке есть специальная стандартная функция rnd(a), значением которой является случайное число от 0 до a. Эта функция обычно называется ***датчиком случайных чисел***. При вызове

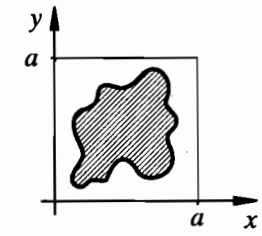


Рис. 111

функции это число каждый раз свое. Если же вызвать функцию много раз подряд, то множество полученных чисел будет равномерно распределено по отрезку $[0, a]$. Пару случайных чисел можно рассматривать как координаты точки на плоскости (координаты «песчинки»). А алгоритм приближенного вычисления площади, использующий n «песчинок», запишется так:

А 145

```
алг вещ площадь (арг цел n, вещ a)
    дано | n — количество "песчинок"
         | a — длина стороны квадрата (рис. 111)
    надо | знач ≈ площадь фигуры
нач цел m
| m :=0
| нц n раз
| | x :=rnd(a); y :=rnd(a)     | получение очередной "песчинки"
| | если точка внутри(x, y)   | подсчет количества
| | | то m :=m+1              | "песчинок", попавших
| | все                       | внутрь фигуры
| кц
| знач :=(m/n)*a*a            | a*a — площадь квадрата
кон
```

Здесь использован вспомогательный алгоритм "точка внутри", который по координатам точки (x, y) должен определять, попала точка внутрь фигуры или нет, — ведь мы предположили, что это мы делать умеем. Для каждой конкретной фигуры нужно составлять свой алгоритм "точка внутри". Взяв в качестве фигуры круг, можно применить алгоритм А145 для приближенного вычисления числа π.

33.7. Вычисление π методом Монте-Карло

Мы знаем, что площадь круга равна πR^2. Рассмотрим круг радиуса 1 с центром в точке (1, 1). Его площадь равна π. С другой стороны, эту же площадь мы можем вычислить приближенно методом Монте-Карло, составив следующий алгоритм "точка внутри":

А 146

```
алг лог точка внутри (арг вещ x, y)
    надо | знач = точка (x, y) внутри круга радиуса 1 с центром (1, 1)
нач
| знач := ((x-1)**2 + (y-1)**2 <= 1)
кон
```


С использованием алгоритмов А145 и А146 приближенное вычисление числа π можно записать в виде команды присваивания: пи := площадь (n, 2), где n — количество «песчинок». Поскольку используются случайные числа, то результаты могут быть разными даже при одном и том же n. В таблице 25 приведены некоторые результаты, полученные при выполнении этой команды авторами учебника.

ТАБЛИЦА 25. **Результаты вычисления числа π**

n	10	100	500	1000	5000
π	4.4	3.28	3.312	3.232	3.1424

ЗАДАЧИ И УПРАЖНЕНИЯ

1. Как с помощью алгоритма "корень" (А142) найти:
   а) корень уравнения $x \sin x - 1 = 0$ на отрезке $[0, 2\pi]$ с точностью до 0,02;
   б) корень кубический из 3 с точностью 0,00001?
2. Как будет работать алгоритм "корень", если: а) функция имеет на заданном отрезке несколько корней; б) значения функции на концах отрезка одного знака; в) функция обращается в нуль точно посередине одного из уменьшающихся отрезков?
3. График функции

А 147

```
алг вещ F (арг вещ x)
нач
| выбор
| | при x >= 1 : знач :=x
| | при x < 1 : знач :=x-2
| все
кон
```

   имеет разрыв в точке $x = 1$. Что произойдет при попытке выполнить для этой функции команду x := корень (0, 2, 0.001)?
4. Какую величину вычисляет алгоритм А144, если функция f может принимать и отрицательные значения?
5. Что получится при выполнении алгоритма А144 для функции $f(x) = |x|$ при $a = -1$, $b = 1$, $n = 5$?
6. Для каких функций алгоритм А144 дает точный ответ?

7. Как вычислить π с использованием алгоритма А144?

8. Автомобиль трогается с места и разгоняется в течение 4 с. Значения ускорения автомобиля в моменты времени $t = 0{,}1$, $t = 0{,}2$, $t = 0{,}3$ и т. д. с шагом 0,1 с измеряются и записываются в таблицу **вещ таб** а[1:40]. Составьте алгоритм, приближенно вычисляющий расстояние, пройденное автомобилем до конца разгона, и скорость в конце разгона.

9. Четыре жука, расположенные в вершинах квадрата со стороной *a*, одновременно начинают ползти по направлению к следующему по часовой стрелке жуку с постоянной скоростью *v*. Составьте алгоритм, который с помощью *Чертежника* рисует приближенно траектории движения жуков за первые *T* секунд после начала движения.

   Указание. Замените траектории жуков ломаными — считайте, что в течение *t* секунд жуки ползут прямо (одно звено ломаной), потом поворачивают, опять *t* секунд ползут прямо и т. д.

10. По образцу алгоритма А146 составьте алгоритмы "точка внутри" для следующих фигур: а) полукруг; б) сектор; в) сегмент.

34 Моделирование и вычислительный эксперимент

34.1. Вычислительный эксперимент

Компьютер — универсальная машина для обработки информации. Поэтому, если мы хотим про какую-то ситуацию что-то узнать (получить информацию!), мы можем попытаться *смоделировать* эту ситуацию на компьютере и получить информацию из анализа не реальной ситуации, а ее информационной модели.

Пусть, например, нас интересует, сколько времени будет падать парашютист с высоты 1000 м, если у него не раскроется парашют. Можно провести настоящий эксперимент — изготовить чучело парашютиста, сбросить его несколько раз с самолета и замерить среднее время падения. Можно, однако, вместо этого «поставить эксперимент на компьютере» — рассчитать время падения парашютиста, пользуясь информационной моделью. В этом параграфе мы покажем, как это делается.

Моделирование (***вычислительный эксперимент***) в некоторых случаях оказывается незаменимым — ведь не всегда можно провести настоящий эксперимент. Нельзя, например, устроить

настоящую ядерную войну, чтобы узнать, как изменится климат, или лишить нашу планету озонового слоя и посмотреть, к чему это приведет.

Конечно, результаты вычислительного эксперимента могут оказаться и несоответствующими реальности, если в информационной модели не будут учтены какие-то важные стороны действительности. Так, в задаче о падении парашютиста для получения результата, соответствующего реальности, надо учитывать сопротивление воздуха, а при расчете запуска ракеты следует иметь в виду уже не только сопротивление, но и его изменение с ростом высоты.

34.2. Метод дискретизации непрерывных процессов

Вернемся к задаче падения парашютиста. Экспериментально установлено, что сила сопротивления воздуха пропорциональна квадрату скорости, а коэффициент зависит от формы тела. Поэтому выражение для ускорения падающего тела имеет вид

$$a = g - kv^2,$$

где k — коэффициент, который определяется массой и формой тела (для человека среднего веса и роста $k \sim 0{,}004$).

Если сопротивление воздуха отсутствует, т. е. $k = 0$, то для изменения высоты со временем и для времени падения из физики известны общие формулы

$$h(t) = h_0 - \frac{gt^2}{2},\ T_{\text{падения}} = \sqrt{\frac{2h_0}{g}}.$$

Если принять $g = 10$, то для начальной высоты $h_0 = 1000$ получится $T_{\text{падения}} = \sqrt{\frac{2 \cdot 1000}{10}} \approx 14$ с.

Если же $k \neq 0$, то точных формул для времени падения не существует и время падения придется вычислять приближенно. Для такого вычисления применяется ***метод дискретизации непрерывных процессов***, который можно объяснить так.

Представьте себе, что мы засняли падение парашютиста на кинопленку. Кинопленка теперь — это некоторая модель реального процесса, по ней можно воссоздать падение, и даже достаточно точно. Реальный процесс падения является непрерывным,

плавным. Кинопленка же состоит из отдельных кадров: на одном — парашютист в одном положении, на соседнем — в следующем, а между этими двумя положениями на пленке ничего нет. Информацию о парашютисте в моменты, не попавшие ни в один из кадров, можно приближенно восстановить, глядя на соседние кадры. Подобного рода замена непрерывного процесса прерывным (принято говорить ***дискретным***) называется ***дискретизацией***.

При дискретизации время обычно разбивается на небольшие интервалы (например, по 0.01 с) и величины, зависящие от времени, вычисляются на компьютере только «для отдельных кадров», т. е. в концах интервалов. В памяти компьютера, подобно кинопленке, оказывается информация лишь об отдельных моментах хода процесса. Скажем, плавно меняющаяся высота h парашютиста над землей будет в памяти компьютера заменена последовательностью значений высот в моменты времени

$$t_0 = 0, \qquad t_1 = 0.01, \qquad t_2 = 0{,}02, \ldots$$

Количество этих моментов зависит от продолжительности моделируемого процесса. Например, если процесс падения моделируется в течение половины минуты, то последним моментом времени будет $t_{3000} = 30$. Чтобы разместить в памяти компьютера значения любой меняющейся в процессе падения величины, удобно использовать таблицу с индексом от 0 до 3000, скажем, значения высот можно разместить в таблицу **вещ таб** h[0, 3000]:

h[0] — значение высоты в момент времени t0 = 0,
h[1] — значение высоты в момент времени t1 = 0.01,
h[2] — значение высоты в момент времени t2 = 0.02 и т. д.

Значения величин в начальный момент времени обычно известны из постановки задачи: в задаче о парашютисте в начальный момент времени высота равна 1000 метров, т. е. h[0] = 1000. Значения величин в остальные моменты времени предстоит вычислить при моделировании. Делается это последовательно, «кадр за кадром». «Глядя» на очередной кадр, компьютер вычисляет, что будет на следующем. Этот процесс разберем на примере задачи о парашютисте.

34.3. Падение парашютиста с высоты 1000 метров с учетом сопротивления воздуха

Будем вести моделирование с интервалом времени dt = 0.01 с в течение 30 секунд. При этом максимальный номер кадра у нас будет 3000. При падении парашютиста кроме времени t и высоты h меняются также скорость v и ускорение a, эти величины при моделировании разместим в таблицах:

вещ таб t[0:3000], h[0:3000], v[0:3000], a[0:3000]

В начальный момент время и скорость считаем равными нулю, высота равна 1000 м, а ускорение — ускорению g свободного падения. Значит, «начальный кадр» можно записать в память компьютера так:

t[0] := 0; h[0] := 1000; v[0] := 0; a[0] = g

Предположим, что в памяти компьютера уже есть «кадр» i, т. е. уже вычислены числа t[i]; h[i], v[i]; a[i].

Вычислить приближенно «следующий кадр», т. е. числа

t[i + 1]; h[i + 1]; v[i + 1]; a[i + 1]

можно по таким формулам:

t[i + 1] := t[i] + dt	кадры отделены друг от друга промежутком времени dt
h[i + 1] := h[i] – v[i]*dt	считаем, что скорость v между кадрами не меняется и равна v[i]
v[i + 1] := v[i] + a[i]*dt	считаем, что ускорение a между кадрами не меняется и равно a[i]
a[i + 1] := g – k*v[i + 1]*v[i + 1]	формула для ускорения с учетом сопротивления воздуха

Зная начальный кадр i = 0 (t = 0), по этим формулам можно вычислить следующий кадр i = 1 (t = 0.01), затем по тем же формулам — кадр i = 2 (t = 0.02) и т. д. Кадр i = 100 дает состояние парашютиста после секунда полета, кадр i = 200 — после двух секунд полета и т. д. Приведенные формулы для t[i+1] и a[i+1] — точные, а для h[i+1] и v[i+1] — приближенные. Чем меньше dt, тем эти приближенные формулы точнее.

Чтобы увидеть результаты моделирования, после вычисления каждого кадра будем печатать на отдельной строке время, высоту и скорость парашютиста:

вывод нс "t =", t[i], "h =", h[i], "v =", v[i]

При падении высота парашютиста над землей уменьшается, поэтому в процессе моделирования h[i] будет уменьшаться. В какой-то момент число h[i] останется еще положительным, а число h[i+1] станет уже отрицательным или нулем. В этот момент моделирование нужно прервать, так как отрицательность h означает, что между моментами времени t_i и t_{i+1} парашютист ударился о землю. Что с ним при этом произойдет — нашими формулами не описывается. Запишем окончательный алгоритм, сделав величины g, k и dt его аргументами:

алг падение (арг вещ g, k, dt) **A 148**

```
    дано | g — ускорение свободного падения
         | k — коэффициент сопротивления воздуха
         | dt >= 0.01 | шаг по времени
    надо | напечатаны значения времени, высоты и скорости парашютиста в моменты времени 0, dt, 2dt, 3dt, ... до момента
         | удара о землю
нач вещ таб t[0: 3000], h[0: 3000], v[0: 3000], a[0: 3000]; цел i
| t[0] :=0; h[0] :=1000; v[0] :=0; a[0] =g
| нц для i от 0 до 2999 пока h[i] > 0
| | вывод нс "t =", t[i], " h =", h[i], " v =", v[i]
| | t[i+1] := t[i] + dt
| | h[i+1] := h[i] - v[i]*dt
| | v[i+1] := v[i] + a[i]*dt
| | a[i+1] := g - k*v[i+1]*v[i+1]
| кц
кон
```

Приведем результаты выполнения алгоритма А148 для $g = 10, k = 0{,}004, dt = 1$:

$t = 0$	$h = 1000$	$v = 0$
$t = 1$	$h = 1000$	$v = 10.0$
$t = 2$	$h = 990$	$v = 19.6$
...		
$t = 22$	$h = 84$	$v = 50.0$
$t = 23$	$h = 34$	$v = 50.0$
$t = 24$	$h = -16$	$v = 50.0$

Последнее отрицательное значение высоты теоретически означает, что тело оказывается под землей на глубине 16 м. Практически это значит, что на 24-й секунде падения тело ударится о землю.

34.4. Сравнение приближенного и точного решений

Если коэффициент сопротивления $k = 0$ (например, при падении в вакууме), то положение и скорость тела в любой момент падения можно вычислить точно: $v(t) = v_0 + gt$, $h(t) = h_0 - v_0 t - \frac{gt^2}{2}$.

В этом случае мы можем сравнить приближенное решение, которое дает алгоритм А148, с точным. Сделаем это для $dt = 1$ и $dt = 0{,}5$:

t	Шаг $dt = 1$		Шаг $dt = 0.5$		Точное решение	
	h	v	h	v	h	v
0	1000	0	1000	0	1000	0
1	1000	10	997.5	10	995	10
2	990	20	985	20	980	20
	...					
13	220	130	187.5	130	155	130
14	90	140	55	140	20	140
15	–50	150	–87.5	150	–125	150

Что можно сказать, глядя на эти числа?

1) Приближенное решение отличается от точного. Причину отличий легко увидеть: при $dt = 1$ в первую секунду (от $t = 0$ до $t = 1$) тело вообще не падает. Это выглядит нелепо, но «что посеешь — то и пожнешь», ведь мы сами при подсчете высоты решили считать скорость неизменной на каждом промежутке времени, а в начале она равна нулю.

2) С уменьшением dt (при переходе от 1 к 0,5) приближенное решение становится ближе к точному.

3) Даже при $dt = 1$ разница между приближенным и точным решениями не так велика: оба они предсказывают, что тело ударится о землю на 15-й секунде падения.

34.5. Выбор шага по времени

Обычно точный ответ приближенного вычисления неизвестен — ведь мы составляем алгоритм как раз для того, чтобы этот ответ узнать! Как же в этом случае оценить точность полученного решения? Исходя из каких соображений устанавливать величину dt?

Существуют строгие математические методы оценки погрешности вычислений, но мы рассмотрим только простейший прием, часто применяемый на практике. Будем уменьшать *dt* и смотреть, что происходит с ответом. Пусть, например, надо найти высоту тела через 20 с после начала падения (h_0 = 1000, k = 0,004). Значения h в момент t = 20 при разных *dt* показаны в таблице 26.

ТАБЛИЦА 26. **Результаты вычислительного эксперимента**

dt	1.0	0.5	0.2	0.1	0.05	0.001
h	184.0	178.3	175.2	174.2	173.7	173.3

Видно, что с уменьшением *dt* разница между соседними результатами становится все меньше (два последних отличаются всего на полметра). Поэтому естественно предположить, что дальнейшее уменьшение *dt* уже не приведет к сильному изменению h, и взять dt = 0,001.

Почему же сразу не взять очень маленькое *dt*, например dt = 0,000001? Дело в том, что чем меньше *dt*, тем больше шагов придется сделать при выполнении алгоритма и тем больше времени уйдет на его выполнение. Если, например, при dt = 0,001 алгоритм А148 выполняется за 1 с, то при dt = 0,000001 он будет выполняться уже больше 15 мин.

Выбор *dt*, впрочем, обычно тоже поручают компьютеру, т. е. составляют алгоритм, который начинает подсчет для какого-то значения *dt*, затем уменьшает *dt*, например вдвое, считает снова и т. д. до тех пор, пока результаты вычислений не станут достаточно близки друг к другу.

34.6. Как обойтись без табличных величин в алгоритме А148

Алгоритм А148 осуществляет вычисление последовательностей «кадров», т. е. четверок чисел:

```
t[0], h[0], v[0], a[0]; начальный кадр
t[1], h[1], v[1], a[1]; кадр номер 1
t[2], h[2], v[2], a[2]; кадр номер 2
t[3], h[3], v[3], a[3]; кадр номер 3
. . . . . .
```


Начальный кадр задан в условии задачи, а вычисление каждого следующего требует знания только предыдущего кадра. Если мы хотим найти лишь последний кадр — перед ударом о землю, то нет необходимости хранить в памяти компьютера все вычисленные ранее кадры, достаточно хранить только последний.

Введем информационную модель одного состояния процесса падения (одного кадра):

М 21

```
вещ t | время, прошедшее с начала падения
вещ h | расстояние до поверхности земли
вещ v | скорость тела
вещ a | ускорение тела
```

Тогда переход к следующему состоянию процесса падения может быть записан так:

```
t :=t+dt
h :=h–v*dt
v :=v+a*dt
a :=g–k*v*v
```

Действительно, если перед выполнением этих команд значения величин t, h, v, a взять из кадра i, то после выполнения команд эти величины будут иметь значения их кадра номер i+1.

Поэтому алгоритм А148 можно переписать без всякого использования таблиц и индексов:

А 149

```
алг падение (арг вещ g, k, dt)
   дано | g — ускорение свободного падения
        | k — коэффициент сопротивления воздуха
        | dt > 0 | шаг по времени
   надо | напечатаны значения времени, высоты и скорости
        | парашютиста в моменты времени 0, dt, 2dt, 3dt, ...
        | до момента удара о землю
нач вещ t, h, v, a | информационная модель состояния процесса
| t :=0; h :=1000; v :=0; a :=g
| нц пока h > 0
| | вывод нс, "t =", t, " h =", h, " v =", v
| | t :=t + dt
| | h :=h – v*dt
| | v :=v + a*dt
| | a := g – k*v*v
| кц
кон
```

Алгоритм А149 лучше алгоритма А148 в двух отношениях:

1) А149 требует гораздо меньше памяти;

2) А149 может работать при любом количестве шагов моделирования, в то время как А148 рассчитан на не более чем заранее фиксированное число шагов.

ЗАДАЧИ И УПРАЖНЕНИЯ

1. Парашютист прыгнул с самолета, летящего со скоростью 180 км/ч на высоте 1300 м, и раскрыл парашют на высоте 600 м. Сопротивление воздуха пропорционально квадрату скорости (k ~ 0,004). Составьте алгоритм, который определяет время падения парашютиста до момента открытия парашюта.

2. В условиях, описанных в упражнении 1, парашютист на высоте 1000 м группируется и коэффициент сопротивления уменьшается с 0,004 до 0,003. Составьте алгоритм, который вычисляет время падения парашютиста до момента открытия парашюта.

3. Составьте алгоритм, аналогичный алгоритму "падение", для расчета колебаний груза на пружинке (ускорение пропорционально величине отклонения от положения равновесия).

4. Шарик подвесили к пружине от школьного динамометра, оттянули вниз от положения равновесия на 1 см и отпустили. Жесткость пружины такова, что в момент отпускания шарика его ускорение под действием силы тяжести и силы упругости пружины равно -4 м/с^2. Составьте алгоритм, который определяет, через сколько секунд шарик поднимется на максимальную высоту.

5. Составьте алгоритм для вычисления координат и скорости мяча, отпущенного на высоте h м над бесконечной наклонной плоскостью, наклоненной под углом a к горизонту, через t секунд после начала движения. Удары упругие.

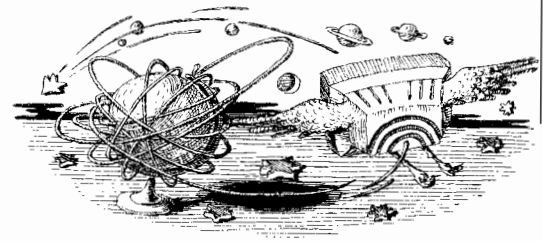
6. Решите упражнение 5, если при каждом отскоке мяча от плоскости модуль его скорости уменьшается на n%.

7. Тело движется по наклонной плоскости под действием силы тяжести. Сила сопротивления пропорциональна скорости тела. Составьте алгоритм, который вычисляет длину пути, пройденного телом за время t от начала движения.

8. На верхнюю ступеньку бесконечной лестницы (ширина ступенек k, высота h) положили мяч и покатили его со скоростью v. Считая мяч материальной точкой, а удары упругими, составьте алгоритм, определяющий номера первых n ступенек, о которые ударится мяч.

ГЛАВА 7

Компьютеры в современном обществе



35 Характеристики современного персонального компьютера

В этом параграфе мы обсудим основные технические характеристики персонального компьютера начала XXI в. и как они сказываются на потребительских свойствах персонального компьютера.

35.1. Основные устройства компьютера

Как уже говорилось в § 2, в состав любого современного персонального компьютера входят три основные части — ***процессор***, ***память*** (***оперативная*** и ***внешняя***) и различные ***внешние устройства***.

Упрощенно работу персонального компьютера можно описать так: процессор выполняет размещенную в памяти программу, оперируя с данными, находящимися в оперативной и во внешней памяти. При этом процессор также управляет внешними устройствами и обменивается информацией с ними, а также с человеком и другими компьютерами.

35.2. Быстродействие процессора, объем оперативной и внешней памяти

Главная часть компьютера — процессор. Он обеспечивает обработку, передачу и прием данных, управление различными устройствами. Процессор имеет собственный, достаточно сложный «язык» и может выполнять фиксированный набор действий-команд (см. § 25). Последовательность команд, записанная на языке процессора и переданная ему для исполнения, называется ***машинной программой***. Технически процессор современного персонального компьютера выполнен в виде ***микропроцессора*** — одной сверхбольшой интегральной схемы, состоящей из нескольких миллионов элементов.

Основная характеристика процессора — ***быстродействие*** (число выполняемых в секунду операций) и ***разрядность*** (количество бит, обрабатываемых процессором за одну операцию). Хотя процессор выполняет разные операции с разной скоростью, удобно говорить про некоторое усредненное быстродействие процессора. Быстродействие современного персонального компьютера выше, чем у самых быстрых в мире компьютеров выпуска 1985 г., и составляет от нескольких сот миллионов до миллиарда операций в секунду. К концу XX в. все персональные компьютеры оснащались 32-разрядными микропроцессорами. Первые персональные компьютеры на 64-разрядных микропроцессорах появились в 2000 г. Микропроцессор разрядности 32 может непосредственно работать с памятью объемом до 2^{32} байт, т. е. около 4 Гбайт. Внешняя память персональных компьютеров уже превзошла этот предел, а оперативная память достигнет его в течение ближайших нескольких лет. Переход к 64-разрядным микропроцессорам позволит без хлопот наращивать память до 2^{64} байт.

Основная характеристика памяти — ее объем. Чем больше объем оперативной памяти, тем больший объем информации процессор способен обрабатывать с максимально возможной скоростью. Чем больше объем внешней памяти, тем больше информации компьютер способен хранить долговременно. Если при выполнении программы (или нескольких программ одновременно) оперативной памяти окажется мало, то процессор начнет использовать для хранения программ и данных не только оперативную память, но и внешнюю. Это замедлит скорость выполнения программ. Обычный объем оперативной памяти современного персонального компьютера от 32 до 128 Мбайт, а объем внешней памяти — от 2 до 16 Гбайт.

35.3. Как можно использовать 2 Гбайта внешней памяти

Пусть на вашем компьютере есть диск объемом 2 Гбайта. Что это значит практически? Как этот объем будет использоваться?

Во-первых, на диске нужно всегда иметь некоторое количество свободного места.

Если в какой-то момент процессору не хватит оперативной памяти, он начнет использовать внешнюю память в помощь оперативной. Для этого желательно зарезервировать 100—200 Мбайт.

Во-вторых, на диске будут размещены жизненно важные программы, скорее всего установленные на вашем компьютере его изготовителем: операционная система и некоторые служебные программы. Можно считать, что все эти программы уместятся в 100—200 Мбайт.

Далее, на вашем компьютере будет установлено некоторое количество программ, с которыми вы работаете постоянно: это могут быть редакторы текстов, электронные таблицы и базы данных, графические программы, электронная почта, программы для работы в Интернете, учебные программы, словари, справочники, игры. Современная программная система может занимать объем от нескольких мегабайт до нескольких десятков мегабайт. Так что десяток ваших любимых программ использует еще несколько сотен мегабайт, скажем 300—600 Мбайт.

Итого на диске уже занято 500—1000 Мбайт. Будем считать, что остался 1 Гбайт. Что можно разместить в оставшейся части диска? В одном гигабайте можно разместить:

- библиотеку из 1—2 тыс. томов (примерно такое количество книг вы в состоянии успеть прочесть в течение вашей жизни; в сети Интернет сейчас свободно доступны тысячи художественных произведений и документов на английском, русском и других мировых языках);
- или фотоальбом из 20—50 тыс. цветных фотографий любительского качества, что на один-два порядка превышает объемы обычных семейных фотоальбомов; если тратить на просмотр одной фотографии 10—20 секунд, то на просмотр 20 тыс. фотографий уйдет несколько месяцев (дешевые любительские цифровые фотокамеры, способные делать такие фотографии, сохранять их в своей памяти, а затем переписывать в память компьютера, появились в продаже в 1996 г. и стоили 300—800 долларов;

сейчас такие фотокамеры становятся массовыми и стоят существенно меньше);

- или свыше 1500 цветных слайдов высочайшего качества (это уже больше похоже на размер семейного альбома);
- или запись речи «телефонного» качества длиной 500—1000 часов (этого хватит, чтобы записать все уроки, на которых вы были в течение последнего года, или все, что вы говорили в течение последних нескольких лет);
- или стереозапись качества дешевого кассетника длиной 50—100 часов (этого достаточно для записи всех альбомов ваших самых любимых групп и исполнителей);
- или 2 часа стереозаписи звука максимально возможного качества;
- или 1—2 часа записи видеофильма любительского качества;
- или 150-секундный фильм высочайшего качества записи.

К концу XX в. объем внешней памяти персонального компьютера возрос до десятков Гбайт, а стоимость — в пересчете на 1 Гбайт — упала до стоимости одной заправки бензином легкового автомобиля (около 40 литров).

Кое-что из перечисленного целесообразно размещать в так называемой однократно записываемой или постоянной (read-only) памяти на оптических дисках (см. п. 35.6).

35.4. Сколько оперативной памяти нужно компьютеру

Для работы операционной системы Windows 95 формально требовалось не менее 8 Мб оперативной памяти, а 16 Мб обеспечивали удовлетворительную работу как самой операционной системы, так и большинства прикладных программ, выпущенных в свет к моменту появления Windows 95.

Годом позже появились программы общего назначения, требующие 24 или 32 Мб оперативной памяти. Для нормальной работы операционных систем Windows NT и Windows 2000 необходимо 32 Мб оперативной памяти. В процессе совершенствования персонального компьютера на нем начинают решаться все более сложные задачи, и требуемый объем оперативной памяти растет из года в год. Рассмотрим, например, задачу компьютерной обработки высококачественных цветных фотографий. При сканировании цветной фотографии размером 4×6 дюймов с разрешением 600 точек на дюйм понадобится около 35 Мб оперативной памяти только для хранения самой фотографии. Как минимум, еще столько же памяти потребуется, когда начнется обработка изображения. Ясно, что для работ с такими фотографиями объем оперативной памяти компьютера должен быть порядка 128 Мб. Всего несколько лет назад такую память имели только дорогие специализированные компьютеры. Сегодня памятью такого объема начинают оснащаться «рядовые» персональные компьютеры.

35.5. Нужен ли нам такой быстрый процессор

Скорость выполнения многих работ на персональном компьютере определяется быстродействием процессора. Скажем, скорость подготовки страницы для печати на струйном принтере пропорциональна скорости работы процессора. Вдвое более быстрый процессор сделает это за вдвое меньшее время. Но есть работы, которые должны быть выполнены с определенной скоростью, иначе за них нечего и браться. Так, если на компьютере установлена шахматная программа и компьютер участвует в турнире, то на всю партию или на первые 40 ходов отводится определенное время. Компьютер, который не в состоянии обдумывать ходы быстро, в таком соревновании участвовать просто не сможет. Другой пример: если нужно проиграть звукозапись, которая в специально закодированном виде читается с лазерного диска, то процессор должен успевать готовить к воспроизведению односекундную запись за время, меньшее одной секунды. Только при этом условии воспроизведение окажется возможным. То же верно и для воспроизведения видеоизображений, генерации изображения на экране в тренажерных программах и играх.

35.6. Постоянная память и однократно-записываемая память на оптических дисках

Если вы хотите хранить на персональном компьютере библиотеку в 1000 томов, то вовсе не обязательно тратить на эту информацию место на магнитном диске. Вы можете купить съемный оптический диск (англ. CD ROM — Compact Disk Read Only Memory), содержащий подобную библиотеку, или купить несколько дисков, скомпоновать библиотеку по своему вкусу и записать ее на однократно записываемый оптический диск. После завершения записи информацию с этого диска можно только читать и нельзя будет изменять.

К середине 90-х гг. емкость такого оптического диска составляла около 650 Мбайт и лучшие дисководы (плейеры) для

таких дисков были способны читать информацию со скоростью, достаточной для показа видеофильмов. Сегодня все персональные компьютеры оснащаются подобными дисководами.

Замечание. Скорость такого дисковода измеряется в забавных единицах. А именно, она сравнивается со скоростью первого устройства подобного рода, скорость которого принята за единицу. Так, например, говорят о скоростях 6×, 10×, 12× и т. д.

К концу XX в. началось широкое распространение нового стандарта записи видеоинформации на оптические диски — формат DVD (Digital Video Disk). В этом новом формате на диск можно записать свыше 10 Гбайт информации.

35.7. Видеосистема персонального компьютера. Растровый принцип вывода графической информации

При выводе графической информации на экран изображение формируется из растровых строчек, состоящих, в свою очередь, из точек, называемых ***пикселами*** (англ. *picture element*). Таким образом, растр — это матрица из точек-пикселов, каждая из которых имеет свой цвет.

Видеосистема включает в себя видеоадаптер и дисплей. Видеоадаптер — это специальная плата, на которой расположены видеопроцессор и видеопамять. Видеоадаптер предназначен для подготовки и хранения растра в видеопамяти и преобразования растра в видеосигналы, подаваемые на дисплей, который и генерирует видимое человеком изображение.

Для каждой точки растра задается ее цвет. Цвет может задаваться одним байтом, в этом случае количество возможных цветов составляет 256, двумя байтами — около 64 тыс. цветов или тремя байтами — 16 млн цветов. Несколько десятков раз в секунду видеоадаптер читает растр строка за строкой и формирует видеосигнал, который подается на дисплей. Последовательно передается цвет первой точки первой строки, потом второй точки первой строки, ..., последней точки первой строки, первой точки второй строки и т. д. Этот процесс однократной передачи изображения от видеоадаптера дисплею называется *передачей кадра*.

Качество изображения определяется в основном тремя параметрами:

1) размером и «глубиной» растра (количество столбцов и строк и количество цветов);

2) частотой передачи кадров;

3) разрешающей способностью экрана дисплея, т. е. расстоянием между точками люминофора на внутренней поверхности экрана.

На 17-дюймовом экране при размере точки изображения 0,23—0,27 мм можно разместить изображение размером 1024×768 или 1280×1024 точек. Если цвет при этом задается 3 байтами, т. е. количество цветов равно 16 млн, то потребуется размер памяти видеоадаптера несколько меньше 4 Мбайт. Этот размер памяти наиболее часто встречается в современных персональных компьютерах. Для растра 1600×1200 глубиной 3 байта понадобится уже 6 Мбайт.

Указанные выше размер экрана 17 дюймов и размер изображения 1280×1024 достаточны для того, чтобы качественно изобразить на экране рядом две страницы текста — такой режим необходим при многих видах работ с текстами на компьютере.

Частоту кадров (количество кадров в секунду) измеряют в герцах. При частотах ниже 60 Гц человек видит мерцание, ощущает дискомфорт, глаза и мозг быстро устают. При частотах между 60 и 72 Гц мерцание уже незаметно, но человек устает быстрее, чем при частотах 80—90 Гц. Дальнейшее увеличение частоты уже не повышает комфорт.

35.8. Принтеры и сканеры в начале XXI века

В современных компьютерах чаще всего применяют два типа принтеров: лазерный и струйный. Вывод изображения на принтер аналогичен выводу на дисплей и также производится по растровому принципу. Основным параметром принтера является разрешение, измеряемое в точках (элементах изображения) на дюйм. Лазерный принтер стоимостью 200—400 долларов печатает черно-белое полутоновое изображение с разрешением 600 точек на дюйм со скоростью 3—6 страниц в минуту. Струйный принтер стоимостью 100—150 долларов делает то же самое со скоростью 1—2 с./мин. Цветной струйный принтер стоимостью 150—300 долларов будет печатать цветное изображение разрешением 600 точек на дюйм со скоростью 0,5—1 с./мин. (Все данные относятся к 2000 году и к формату листа А4 — это стандартный формат писчей бумаги.)

Многие персональные компьютеры сегодня оснащены цветными сканерами формата А4. Сканер стоимостью 100—300 долларов позволит вводить в компьютер цветное изображение

с разрешением 600 точек на дюйм «глубиной» 3 байта на точку за 1—2 минуты.

Сканер может быть оснащен программами оптического распознавания печатных текстов. Комплекс сканер—компьютер—принтер можно использовать как относительно медленное копирующее устройство.

35.9. Аудиоадаптер

Звук представляет собой колебание воздуха. Амплитуда этого колебания непрерывно меняется со временем. По своей природе звук является непрерывным процессом. Чтобы кодировать звук, надо превратить его в дискретный сигнал, последовательность нулей и единиц. Для этого можно измерять амплитуду звука несколько тысяч раз в секунду, скажем 44 032 раз в секунду, и результат каждого измерения записывать в виде 16-разрядного двоичного числа. Аудиоадаптер в основном служит для преобразования звука из непрерывной формы в дискретную при записи звука и из дискретной, компьютерной формы в аналоговую — при воспроизведении. Современный персональный компьютер может быть оснащен аудиоадаптером, способным проводить указанные преобразования двухканального стереозвука. Дальнейшее увеличение частоты или разрядности не приводит к улучшению качества восприятия звука человеком.

Все выпущенные в конце XX в. аудиоадаптеры имеют один и тот же набор базовых возможностей, достаточный для большинства применений.

Кроме методов кодирования произвольного звука есть специальные методы кодирования музыки. В середине 80-х гг. ведущие производители электронных музыкальных синтезаторов и компьютеров договорились о системе команд универсального синтезатора, способного воспроизводить звуки громадного количества существующих музыкальных инструментов. Такой стандарт называется MIDI (англ. *Musical Instrument Digital Interface*). Запись музыкального произведения в формате MIDI есть не что иное, как программа игры воображаемого исполнителя на воображаемом музыкальном инструменте — MIDI-синтезаторе. Превращение такой программы в аналоговый сигнал проводится специальной микросхемой, установленной на плате аудиоадаптера.

35.10. Персональный компьютер и охрана здоровья

Государственные стандарты России называют несколько опасных и вредных факторов для пользователя персонального компьютера. Среди них:

— повышенный уровень шума на рабочем месте;

— повышенная или пониженная ионизация воздуха;

— повышенный уровень электромагнитных излучений и напряженности электрического и магнитного полей;

— повышенный уровень ультрафиолетовой и инфракрасной радиации;

— повышенная пульсация светового потока.

Самый шумный элемент персонального компьютера — вентилятор охлаждения системного блока. Исправный вентилятор не производит опасных для здоровья шумов.

Остальные вредные факторы на 99% создаются дисплеем. Как минимум, дисплей должен удовлетворять российскому стандарту, а еще лучше — шведскому стандарту MPR-II.

35.11. Связь с другими компьютерами

Для персонального компьютера индивидуального пользования основным техническим средством связи с другими компьютерами пока остается модем, который позволяет передавать любую информацию по телефонным линиям. Максимально возможная скорость передачи информации по обычным телефонным линиям через стандартное оборудование АТС составляет около 30 Кбит в секунду, или около 3 Кбайт в секунду. Лучшие модемы обеспечивают, быть может, более надежную связь, но скорость передачи уже упакованной, уплотненной информации не может превысить теоретический предел 3 Кбайт/с. За один час модем способен передать около 10 Мбайт информации, или около 5000 тыс. книжных страниц текста.

Два владельца персональных компьютеров, предварительно договорившись, могут обмениваться информацией по телефону через модемы с указанной скоростью. Любой владелец персонального компьютера, имеющий телефон и модем, может поставить свой компьютер «на дежурство», т. е. в режим приема запросов от других компьютеров через модем. Дежурный компьютер будет принимать звонки от всех желающих или только от зарегистрированных ранее на этом компьютере пользователей. Такой компьютер вместе с размещенной на нем информацией

называется BBS — *Bulletin Board System* — или электронная доска объявлений. Подобные компьютеры есть во многих крупных городах России.

Наконец, во многих городах России имеются компьютеры сети Интернет, с которыми зарегистрированные пользователи могут соединиться через модем и отправить и принять электронную почту или связаться с мировой сетью. Подробнее об этом рассказано в следующем параграфе.

36 Мировые информационные сети

36.1. Информационные сети в докомпьютерную эпоху

В истории человечества вторая половина XX в. — это время возникновения и расцвета общепланетарных информационных систем, образования единого информационного пространства, охватывающего всю планету.

Первыми вошли в практику *широковещательные* системы распространения информации: радиопередачи и телепередачи. Радиопередачи на коротких волнах от одной радиостанции могут приниматься за тысячи километров от передающего центра, слышны на громадных территориях. Несколько радиопередающих центров в состоянии обеспечить прием радиосигналов с помощью обычного бытового радиоприемника практически на всем земном шаре.

Телепередачи удается принимать лишь в радиусе 100—150 км от передающего центра, но в большинстве стран в настоящее время действуют общенациональные системы телевещания. Такая общенациональная система позволяет транслировать передачи из одного центра на множество локальных передатчиков или центров кабельного телевещания и тем самым делает эти передачи доступными всему населению страны. Все эти общенациональные сети способны обмениваться информацией между собой (как правило, с использованием спутников), и тем самым имеется принципиальная возможность организации трансляции одной передачи на весь земной шар. И это не чисто теоретическая возможность. По оценкам экспертов, прямую трансляцию церемонии открытия летних Олимпийских игр в Атланте в 1996 г. смотрело более миллиарда человек.

Если к нам вдруг прилетят инопланетяне, то понадобится не более получаса, чтобы проинформировать об этом две трети населения нашей планеты: существует общепланетарная сеть широковещания.

Широковещательные системы предполагают централизованную подготовку и одностороннюю передачу информации: получатель информации может лишь настроиться на одну из передаваемых программ, но не имеет возможности «заказать» интересующую его информацию, оперативно регулировать форму или скорость подачи информации. Рядовой гражданин только принимает, «потребляет» информацию. Возможности рядовых граждан и мелких предприятий «вещать», передавать информацию, используя традиционные широковещательные сети, весьма ограничены (доступны только в виде платных объявлений).

Наряду с широковещанием возникла и другая планетарная информационная система: телефонная связь. Интересно, что после изобретения телефонии в Америке в конце XIX в. первые идеи коммерческого использования этого изобретения лежали в области широковещания: предполагалось передавать из единого центра многим потребителям (подписчикам) музыкальные программы, новости, погоду и важные официальные сообщения.

Но эту миссию быстро взяли на себя локальные радиостанции. Стало ясно, что основное направление развития телефонии вовсе не централизованное широковещание, а обеспечение децентрализованной, демократичной системы связи: каждый может связаться с кем хочет и передать что хочет. В информатике такая система связи называется ***каждый с каждым*** или ***из точки в точку*** (англ. *point to point*). К концу XX в. локальная телефонная связь в индустриально развитых странах стала повсеместно доступной и дешевой. Связь за пределами одной локальной телефонной сети хотя и сделалась столь же доступной, как и локальная связь, но осталась относительно дорогой.

36.2. Какого рода сигналы передаются по телефонной сети

Телефонные сети с момента своего возникновения и до последнего десятилетия XX в. были ориентированы на передачу звука, в основном речи. С инженерной точки зрения эти сети предназначались для передачи аналоговых сигналов, непрерывных колебаний частотой примерно до 20 Кгц (20 тыс. колебаний в секунду). Для высококачественной передачи речи достаточно частот до 6—8 Кгц, а частоты выше 16 Кгц человеческое ухо вообще не способно воспринимать, так что более высокие частоты для телефонной

с разрешением 600 точек на дюйм «глубиной» 3 байта на точку за 1—2 минуты.

Сканер может быть оснащен программами оптического распознавания печатных текстов. Комплекс сканер—компьютер—принтер можно использовать как относительно медленное копирующее устройство.

35.9. Аудиоадаптер

Звук представляет собой колебание воздуха. Амплитуда этого колебания непрерывно меняется со временем. По своей природе звук является непрерывным процессом. Чтобы кодировать звук, надо превратить его в дискретный сигнал, последовательность нулей и единиц. Для этого можно измерять амплитуду звука несколько тысяч раз в секунду, скажем 44 032 раз в секунду, и результат каждого измерения записывать в виде 16-разрядного двоичного числа. Аудиоадаптер в основном служит для преобразования звука из непрерывной формы в дискретную при записи звука и из дискретной, компьютерной формы в аналоговую — при воспроизведении. Современный персональный компьютер может быть оснащен аудиоадаптером, способным проводить указанные преобразования двухканального стереозвука. Дальнейшее увеличение частоты или разрядности не приводит к улучшению качества восприятия звука человеком.

Все выпущенные в конце XX в. аудиоадаптеры имеют один и тот же набор базовых возможностей, достаточный для большинства применений.

Кроме методов кодирования произвольного звука есть специальные методы кодирования музыки. В середине 80-х гг. ведущие производители электронных музыкальных синтезаторов и компьютеров договорились о системе команд универсального синтезатора, способного воспроизводить звуки громадного количества существующих музыкальных инструментов. Такой стандарт называется MIDI (англ. *Musical Instrument Digital Interface*). Запись музыкального произведения в формате MIDI есть не что иное, как программа игры воображаемого исполнителя на воображаемом музыкальном инструменте — MIDI-синтезаторе. Превращение такой программы в аналоговый сигнал проводится специальной микросхемой, установленной на плате аудиоадаптера.

35.10. Персональный компьютер и охрана здоровья

Государственные стандарты России называют несколько опасных и вредных факторов для пользователя персонального компьютера. Среди них:

— повышенный уровень шума на рабочем месте;

— повышенная или пониженная ионизация воздуха;

— повышенный уровень электромагнитных излучений и напряженности электрического и магнитного полей;

— повышенный уровень ультрафиолетовой и инфракрасной радиации;

— повышенная пульсация светового потока.

Самый шумный элемент персонального компьютера — вентилятор охлаждения системного блока. Исправный вентилятор не производит опасных для здоровья шумов.

Остальные вредные факторы на 99% создаются дисплеем. Как минимум, дисплей должен удовлетворять российскому стандарту, а еще лучше — шведскому стандарту MPR-II.

35.11. Связь с другими компьютерами

Для персонального компьютера индивидуального пользования основным техническим средством связи с другими компьютерами пока остается модем, который позволяет передавать любую информацию по телефонным линиям. Максимально возможная скорость передачи информации по обычным телефонным линиям через стандартное оборудование АТС составляет около 30 Кбит в секунду, или около 3 Кбайт в секунду. Лучшие модемы обеспечивают, быть может, более надежную связь, но скорость передачи уже упакованной, уплотненной информации не может превысить теоретический предел 3 Кбайт/с. За один час модем способен передать около 10 Мбайт информации, или около 5000 тыс. книжных страниц текста.

Два владельца персональных компьютеров, предварительно договорившись, могут обмениваться информацией по телефону через модемы с указанной скоростью. Любой владелец персонального компьютера, имеющий телефон и модем, может поставить свой компьютер «на дежурство», т. е. в режим приема запросов от других компьютеров через модем. Дежурный компьютер будет принимать звонки от всех желающих или только от зарегистрированных ранее на этом компьютере пользователей. Такой компьютер вместе с размещенной на нем информацией

называется BBS — *Bulletin Board System* — или электронная доска объявлений. Подобные компьютеры есть во многих крупных городах России.

Наконец, во многих городах России имеются компьютеры сети Интернет, с которыми зарегистрированные пользователи могут соединиться через модем и отправить и принять электронную почту или связаться с мировой сетью. Подробнее об этом рассказано в следующем параграфе.

36 Мировые информационные сети

36.1. Информационные сети в докомпьютерную эпоху

В истории человечества вторая половина XX в. — это время возникновения и расцвета общепланетарных информационных систем, образования единого информационного пространства, охватывающего всю планету.

Первыми вошли в практику *широковещательные* системы распространения информации: радиопередачи и телепередачи. Радиопередачи на коротких волнах от одной радиостанции могут приниматься за тысячи километров от передающего центра, слышны на громадных территориях. Несколько радиопередающих центров в состоянии обеспечить прием радиосигналов с помощью обычного бытового радиоприемника практически на всем земном шаре.

Телепередачи удается принимать лишь в радиусе 100—150 км от передающего центра, но в большинстве стран в настоящее время действуют общенациональные системы телевещания. Такая общенациональная система позволяет транслировать передачи из одного центра на множество локальных передатчиков или центров кабельного телевещания и тем самым делает эти передачи доступными всему населению страны. Все эти общенациональные сети способны обмениваться информацией между собой (как правило, с использованием спутников), и тем самым имеется принципиальная возможность организации трансляции одной передачи на весь земной шар. И это не чисто теоретическая возможность. По оценкам экспертов, прямую трансляцию церемонии открытия летних Олимпийских игр в Атланте в 1996 г. смотрело более миллиарда человек.

Если к нам вдруг прилетят инопланетяне, то понадобится не более получаса, чтобы проинформировать об этом две трети населения нашей планеты: существует общепланетарная сеть широковещания.

Широковещательные системы предполагают централизованную подготовку и одностороннюю передачу информации: получатель информации может лишь настроиться на одну из передаваемых программ, но не имеет возможности «заказать» интересующую его информацию, оперативно регулировать форму или скорость подачи информации. Рядовой гражданин только принимает, «потребляет» информацию. Возможности рядовых граждан и мелких предприятий «вещать», передавать информацию, используя традиционные широковещательные сети, весьма ограничены (доступны только в виде платных объявлений).

Наряду с широковещанием возникла и другая планетарная информационная система: телефонная связь. Интересно, что после изобретения телефонии в Америке в конце XIX в. первые идеи коммерческого использования этого изобретения лежали в области широковещания: предполагалось передавать из единого центра многим потребителям (подписчикам) музыкальные программы, новости, погоду и важные официальные сообщения.

Но эту миссию быстро взяли на себя локальные радиостанции. Стало ясно, что основное направление развития телефонии вовсе не централизованное широковещание, а обеспечение децентрализованной, демократичной системы связи: каждый может связаться с кем хочет и передать что хочет. В информатике такая система связи называется ***каждый с каждым*** или ***из точки в точку*** (англ. *point to point*). К концу XX в. локальная телефонная связь в индустриально развитых странах стала повсеместно доступной и дешевой. Связь за пределами одной локальной телефонной сети хотя и сделалась столь же доступной, как и локальная связь, но осталась относительно дорогой.

36.2. Какого рода сигналы передаются по телефонной сети

Телефонные сети с момента своего возникновения и до последнего десятилетия XX в. были ориентированы на передачу звука, в основном речи. С инженерной точки зрения эти сети предназначались для передачи аналоговых сигналов, непрерывных колебаний частотой примерно до 20 Кгц (20 тыс. колебаний в секунду). Для высококачественной передачи речи достаточно частот до 6—8 Кгц, а частоты выше 16 Кгц человеческое ухо вообще не способно воспринимать, так что более высокие частоты для телефонной

связи не были нужны. С точки зрения информатики телефонные сети были рассчитаны на передачу небольших объемов информации — несколько десятков тысяч бит в секунду.

36.3. Передача цифровой информации по телефонным линиям

В последнее десятилетие телефонные сети стали использоваться для передачи изображений: появились факсимильные аппараты (факсы). И наконец, появились модемы — устройства для преобразования цифровой, компьютерной информации в звуковые колебания и обратно. (Модем = ***мо***дулятор + ***дем***одулятор.) Благодаря распространению модемов по телефонным сетям передается все больше компьютерной, ***цифровой*** информации.

Передача цифровой информации от одного компьютера к другому через модемы и телефонную сеть с инженерной точки зрения выглядит так. Вначале цифровая информация от компьютера-источника поступает на передающий ***модем***. Это устройство генерирует аналоговый звуковой сигнал, который ***модулируется*** цифровой информацией. Аналоговый сигнал по телефонному кабелю (медным проводам длиной до нескольких километров) поступает на телефонную станцию. Но современные телефонные станции — это фактически специализированные компьютеры, работающие с дискретными, цифровыми сигналами. Поэтому на телефонной станции звуковой сигнал немедленно преобразуется в цифровую форму. Далее этот цифровой сигнал путешествует от одной телефонной станции к другой и, наконец, попадает на телефонную станцию получателя. Там он преобразуется в звуковой, аналоговый сигнал и по телефонному кабелю отправляется к месту назначения, где принимающий модем преобразует (***демодулирует***) его в цифровой и посылает компьютеру-получателю.

Таким образом, в процессе передачи сигнал претерпевает массу преобразований:

цифра → аналог → цифра → аналог → цифра

Согласитесь, что такая схема выглядит очень странно. Это объясняется тем, что она основана на двух разных технологиях: старой аналоговой и новой цифровой. Вместе с тем использование готовой телефонной инфраструктуры для передачи компьютерной информации позволяет ускорить процесс создания Всемирной сети, охватывающей все рабочие места и все домашние хозяйства в мире.

36.4. Скорости передачи цифровой информации по телефонным линиям

Теоретический предел скорости передачи информации по приведенной выше схеме — около 30 тыс. бит в секунду[1]. Куда естественнее было бы передавать сигнал в цифровом виде без всяких преобразований в аналоговый, звуковой сигнал. И такие системы так называемой *цифровой телефонной связи*, способные передавать в цифровом виде как звуковую, так и любую другую информацию, стали появляться к концу XX столетия. Самая распространенная из подобных систем, ISDN, в состоянии передавать от 128 тыс. до 2 млн бит информации в секунду (ISDN — англ. *Integrated Service Digital Network*). При использовании ISDN для обычной речевой связи нужен специальный телефонный аппарат, превращающий звуковые колебания в цифровой сигнал и обратно. Будущее ISDN (и других технологий, использующих существующие медные телефонные кабели) туманно. С инженерной точки зрения более перспективна передача информации в каждый дом по оптическим кабелям, которые применяются в системах кабельного телевидения. Но телефонные линии и телефонные станции уже существуют, а готовой оптоволоконной сети нет, и ее развитие потребует долгого времени и больших расходов.

36.5. Компьютер + телефонная связь = = новые возможности

Как только персональные компьютеры и модемы стали доступны рядовым гражданам, появились новые, децентрализованные, методы распространения информации. Теперь любые два владельца персональных компьютеров получили возможность посылать друг другу любую информацию, т. е., попросту говоря, обмениваться компьютерными файлами. Насколько эффектив-

[1] Если компьютер — источник информации — установлен на телефонной станции или соединен со станцией специальной *выделенной телефонной линией*, то преобразования сигналов в звуковые можно вообще избежать. Правда, для этого принимающий модем должен уметь принимать цифровые сигналы. Такие модемы начали производиться в 1996 г. Они способны обеспечить прием информации на компьютер-получатель со скоростью до 56 тыс. бит в секунду. Наконец, при установке на телефонной станции специального оборудования при определенных условиях возможна передача информации на компьютер-получатель со скоростью более 1 Мбит в секунду.

на такая связь? Предположим, что модемы на передающем и принимающем компьютерах и оборудование телефонных станций обеспечивают скорость передачи 14 400 бит в секунду. Тогда пересылка страницы текста займет 2—3 секунды, пересылка любительской цветной фотографии — 10—30 секунд, а пересылка программной системы объемом 1 Мбайт —5—10 минут.

Более того, при установке подходящего программного обеспечения каталог файлов одного компьютера можно сделать видимым на другом, чтобы с удаленного компьютера видеть содержимое этого каталога, записывать и читать файлы и т. д.

Наконец, на домашнем компьютере нетрудно организовать ***электронный журнал объявлений***. Для этого на компьютере нужно установить специальную программу и подсоединить его через модем к телефонной сети. Такой компьютер, а также и установленная на нем программа называется ***сервером***. Хозяин сервера зарегистрирует какое-то количество «подписчиков», сообщит каждому из них имя и пароль для связи с сервером. Группа может состоять, например, из учеников одного класса, или из любителей компьютерных игр, или из владельцев мотоциклов «Ява». Каждый подписчик может позвонить на сервер и прочесть информацию, оставленную другими членами группы; поместить информацию, интересную другим членам группы; поместить информацию, предназначенную для конкретного человека; прочесть информацию, адресованную лично звонящему.

Организация такой системы распространения и обмена информацией не требует ничего, кроме персональных компьютеров с модемами и телефонов у каждого участника. Но с развитием мировой сети Интернет такие системы уходят в прошлое.

36.6. Компьютеризация планеты Земля на рубеже III тысячелетия

Первые ЭВМ появились около 50 лет назад. За это время технология производства компьютеров и программ для них прошла большой путь. Бесспорно, компьютеры изменили общество. Финансы, делопроизводство, промышленное производство, наука — все это сегодня немыслимо без компьютеров, всюду при обработке информации используются компьютеры. Вместе с тем до недавнего времени рядом с компьютерными технологиями хранения, обработки и передачи информации существовали и некомпьютерные. Кроме того, повседневная жизнь, быт оказались меньше затронуты новыми компьютерными технологиями,

чем производственная сфера. В конце XX в. ситуация начала резко меняться. Описать эту ситуацию можно так:

— возникают мировые информационные сети и их конгломераты, охватывающие подавляющее большинство рабочих мест и домашних хозяйств;

— накопленная человечеством к концу XX в. информация целиком переводится в компьютерную форму и поступает на бессрочное хранение в мировые информационные сети;

— вся вновь генерируемая человечеством информация производится посредством компьютеров и технологически может быть сделана доступной на планетарном уровне;

— процессы планетарной информатизации развиваются децентрализованно, на любом этапе сосуществуют и конкурируют разные подходы, технологии, стандарты, протоколы и т. д.;

— при описании новых компьютерных технологий, как правило, применяется английский язык; число людей, использующих его как язык технического описания, начинает превосходить число тех, для кого английский является родным языком;

— подавляющее большинство членов общества XXI в., от школьников до пенсионеров, будет каждодневно взаимодействовать с информационными сетями от локального до планетарного уровня.

Из этого вытекает прогноз на будущее:

— объем оперативно доступной индивидууму информации увеличится на много порядков;

— формы представления и методы доступа к информации будут отличаться большим разнообразием;

— умение работать с существующими информационными структурами и быстро осваивать новые во многом определяет социальный статус индивидуума, эти умения придется поддерживать и обновлять на протяжении всей жизни.

36.7. История возникновения Интернета

Как это ни удивительно, толчком к созданию Интернета послужил запуск в 1957 г. первого искусственного спутника Земли в СССР. Немедленно после этого по указанию президента Д. Эйзенхауэра в недрах министерства обороны США было организовано Агентство перспективных исследовательских проектов (англ. *Advanced Research Project Agency* — ARPA). В 1962 г. это агентство поставило своей целью создание компьютерных компонент и программного обеспечения, позволяющих связать

между собой удаленные компьютеры. Так возникла сеть ARPANET — прародительница Интернета.

В основу этой сети была положена новая для того времени идея: для повышения надежности связи отправлять информацию от компьютера к компьютеру не непрерывно, а мелкими порциями — ***пакетами***. Не получив очередной пакет или получив искаженный пакет, компьютер-получатель мог потребовать от компьютера-отправителя послать пакет заново. Эта идея оказалась весьма удачной. Прошло 10 лет, и на ее основе были обеспечены самые разные виды связи между компьютерами: пересылка файлов, запуск программ, электронная почта и т. д. Наконец, спустя еще 10 лет, в 1982 г. был стандартизован протокол пакетной передачи информации (так называемый ***протокол TCP/IP***), на базе которого работает сегодняшний Интернет.

По этому протоколу было связано между собой

1 тыс. компьютеров в 1984 г.,

10 тыс. компьютеров в 1987 г.,

100 тыс. компьютеров в 1989 г.,

1 млн компьютеров в 1992 г.,

13 млн компьютеров в 1996 г.

На 1 января 2000 г. Интернет связывал 70 млн компьютеров.

Сердце сети Интернет составляют несколько десятков мощных компьютеров, соединенных между собой линиями связи, передающими информацию со скоростью 200 млн бит в секунду. К этим компьютерам, в свою очередь, подсоединяются линиями связи меньшей пропускной способности тысячи других компьютеров. К тем — сотни тысяч менее мощных компьютеров. И наконец, миллионы компьютеров подсоединены к Интернету через специальные или обычные телефонные каналы.

36.8. Виды информации, передаваемой через Интернет

Через Интернет можно передавать любую информацию, разбивая ее на пакеты. Наибольшее распространение на настоящий день получили три вида услуг:

- передача файлов

(FTP-протокол — англ. *File Transfer Protocol*);

- электронная почта

(SMTP-протокол — *Simple Mail Transfer Protocol*);

- передача гипертекстов

(HTTP-протокол — *HyperText Transfer Protocol*).

Новые виды услуг рождаются в Интернете практически каждый день. Например, в конце 90-х гг. появилась возможность слушать радиопередачи. Звук в цифровом виде передается на компьютер слушателя и воспроизводится с помощью аудиосистемы компьютера. Поскольку передача идет через Интернет, оказывается совершенно не важно, как далеко находится радиостанция. Российские радиостанции, вещающие в Интернет (в их число входит, например, известная радиостанция «Эхо Москвы»), одинаково хорошо слышны по всей территории России и мира, где есть доступ к сети. В последнее время таким же образом можно смотреть и телепередачи.

36.9. Передача информации пакетами

Логически в основу Интернета положена идея равноправия всех компьютеров в сети. Каждый компьютер в Интернете имеет уникальный 32-битный адрес, так называемый ***IP-адрес*** (англ. *Internet Protocol address*). Количество возможных адресов — около 4 млрд[1]).

Распределением этих ***числовых*** адресов и ***символических*** (словесных) имен организаций занимается американская ассоциация InterNIC. Это, пожалуй, единственная часть Интернета, которая управляется централизованно.

Каждый пакет информации, посылаемый в Интернет, содержит служебную информацию, позволяющую доставить его к месту назначения.

Эта информация включает, в частности:

- IP-адрес получателях;
- IP-адрес отправителя;
- время отправки;
- «срок годности» пакета;
- информацию о том, как данный пакет должен быть соединен с другими пакетами для получения исходного сообщения.

Набор правил, описывающих, как разбить передаваемую информацию на пакеты, как передавать эти пакеты от компьютера к компьютеру и как собрать из них посланную информацию, называется TCP/IP-протоколом (англ. *Transmission Control Protocol /Internet Protocol*).

[1] Хотя 4 млрд и очень большое число, совершенно ясно, что в ближайшее время адресов перестанет хватать, и уже идет работа по новому стандарту (IPv6 — *Interner Protocol, version 6*), в котором адресов будет намного больше.

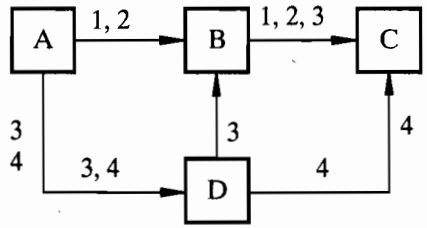
Пакет аналогичен письму, посылаемому по почте: само письмо находится в конверте, а на конверте есть адрес получателя и адрес отправителя, дата отправки, способ доставки (АВИА, заказное и т. д.). TCP/IP-протокол аналогичен набору правил почтовой службы: что должно быть написано на конверте и в каком месте, как выбирается маршрут и метод доставки письма и т. д.

Письмо, посланное по почте, проходит на своем пути несколько сортировочных станций. То же происходит и с пакетами информации в Интернете. На пути к месту назначения пакет обычно проходит 15—20 компьютеров. Каждый из них старается послать пакет куда-то поближе к месту назначения, выбирая наименее загруженный в данное время маршрут. Так что после разбиения посылаемой информации на пакеты и их отправки пакеты могут идти до получателя разными маршрутами и прибывают не обязательно в том порядке, в котором они посылались.

Если промежуточный компьютер не в состоянии отправить пакет получателю или срок годности пакета истек, этот компьютер может уведомить отправителя, поскольку адрес отправителя содержится в пакете.

Разберем это на примере (рис. 112). Предположим, требуется послать файл от компьютера A к компьютеру C. Прямой связи между A и C нет, нужно использовать либо B, либо D. Перед началом пересылки файл разбивается на 4 пакета, каждый из которых содержит адрес отправителя и получателя. Первые два пакета посылаются от A к B, и далее B отправляет их к C. Но в какой-то момент канал между A и B оказывается перегруженным и A отсылает пакеты 3 и 4 к D. Получив пакет 3, D обнаруживает, что канал от D к C временно неисправен. Поэтому пакет 3 посылается на B, который переправляет этот пакет к C. Наконец, к моменту прихода на компьютер D пакета 4 с каналом между D и C уже все в порядке и пакет 4 посылается к C. В итоге порядок приема пакетов на компьютере C оказывается 1, 2, 4, 3. Используя служебную информацию в каждом пакете, компьютер C восстанавливает последовательность пакетов и собирает из них передаваемый файл.

Порядок посылки: 1, 2, 3, 4



Порядок приема: 1, 2, 4, 3

Рис. 112

36.10. Символические адреса

Цифровой адрес в Интернете (IP-адрес) — это четыре числа, от 0 до 255 каждое. При записи эти 4 числа отделяются друг от друга точками, например 155.250.100.22. Человеку цифровые адреса неудобны, поэтому кроме них используются и символические, например niisi.ras.ru, ford.com. Символические адреса имеют не только отдельные компьютеры, но и организации (точнее, локальные сети ЭВМ в пределах одной организации). Символические адреса в разных организациях должны быть разными, иначе возникнет путаница, поэтому они выделяются и регистрируются централизованно, только в ассоциации InterNIC, которая обслуживает всю планету.

Поскольку при любой пересылке информации в Интернете используются числовые IP-адреса, символические адреса перед использованием должны быть преобразованы в числовые: каждому символическому адресу однозначно соответствует числовой. База данных, содержащая информацию о всех символических адресах, имеет объем сотни мегабайт и меняется ежедневно, так что поддерживать ее на каждом компьютере Интернета нереально. Эту базу поддерживают на специальных компьютерах, называемых серверами имен (англ. DNS — *Domain Name Server*). И на каждом компьютере в Интернете достаточно помнить адрес одного ближайшего сервера имен (или для повышения надежности двух-трех). После этого для перевода символического имени в числовое можно послать запрос на сервер имен.

37 Информационные возможности Интернета

37.1. Возникновение World Wide Web — WWW

В середине 90-х гг. в жизни человечества произошло событие планетарного масштаба, важность которого трудно переоценить. Мировая компьютерная сеть Интернет перестала быть достоянием узкого круга ученых и инженеров и за пару лет стала доступной десяткам миллионов рядовых граждан развитых и развивающихся стран. В газетных статьях, телевизионной рекла-

ме, на визитных карточках и в записных книжках запестрели новые магические сочетания:

http://www.gov.ru/
http://www.moscownews.ru/
http://www.64.ru/
http://www.aquarium.ru/
http://www.glasnet.ru/
http://www.harvard.edu/
http://www.msu.ru/
http://www.ford.com/ и
http://www.niisi.msk.ru/~agk

Это адреса так называемых *головных страниц* или *WWW страниц* различных фирм и организаций или частных лиц.

На компьютере, имеющем выход в Интернет, эти адреса позволяют мгновенно

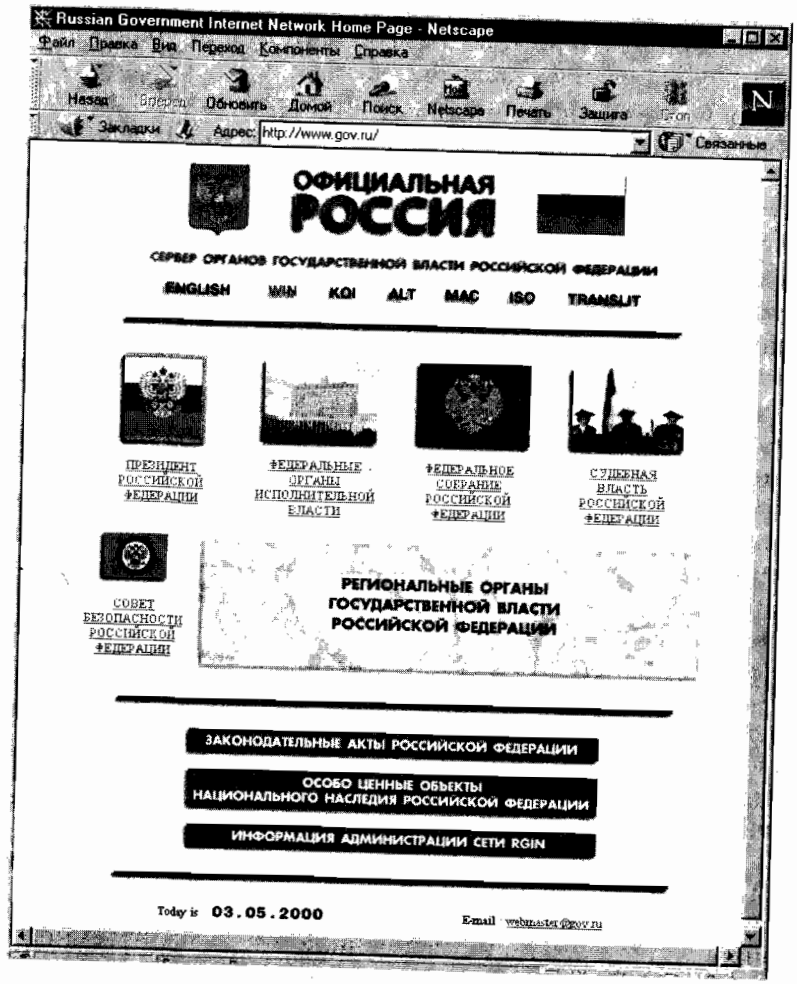
- узнать последние новости исполнительной, законодательной и судебной ветвей власти России;
- посмотреть последний номер газеты «Московские новости» или шахматного журнала «64»;
- узнать у Бориса Гребенщикова новости группы «Аквариум» и даже послушать последний сингл Гребенщикова;
- узнать, какие услуги и по каким ценам предоставляет популярная в России фирма «Гласнет»;
- узнать, как поступить в Гарвардский университет или Московский государственный университет;
- увидеть фотографии всех моделей фирмы «Форд» в текущем году и узнать, чем она собирается удивить покупателей;
- а также ознакомиться с той информацией о сотруднике Института Системных Исследований Российской Академии Наук (НИИСИ) Анатолии Кушниренко, которую тот пожелал сделать доступной всему миру (биография, фото на фоне МГУ, научные планы, сведения о спецкурсах и семинарах, читаемых на механико-математическом факультете МГУ, и т. д.).

До появления WWW Интернет был подобен первому персональному компьютеру фирмы IBM с алфавитно-цифровым экраном и операционной системой MS DOS. Чтобы хоть что-нибудь сделать, требовалось набрать несколько команд, имен файлов, каталогов, ключей и т. д. И все эти команды нужно было помнить наизусть и набирать без единой ошибки. Запомнить все это и, главное, получать удовольствие от подобного рода работы были способны немногие.

После появления технологии WWW Интернет стал похож на мультимедийный компьютер с цветной высококачественной графикой, стереозвуком и современной многооконной многозадачной операционной системой, где все действия совершаются

мгновенно двумя-тремя щелчками мышкой, а если вы что-то забыли или никогда не знали, то один-два щелчка — и вы видите на экране справочный файл с цветными иллюстрациями и даже мультфильмами, в котором объясняется, что и как нужно делать в интересующей вас ситуации.

Основная идея WWW грандиозна: представить всю информацию, доступную в Интернете, текстовую, графическую, да и любую другую, как единый общепланетарный объект, называемый Всемирной Паутиной (английское словосочетание World Wide Web буквально может быть переведено как «сеть, охватывающая весь мир»). Профессионалы часто говорят просто Web,



или WWW. Еще правильнее было бы назвать WWW Всемирным Гипертекстом, поскольку именно понятие гипертекста положено в основу WWW.

37.2. Всемирный гипертекст: URL и HTTP

Слово ***гипертекст*** состоит из двух частей: приставки *гипер-* и корня *текст*. И действительно, в основе гипертекста лежит текст, но текст этот устроен сложнее обычного, напечатанного в книге, и работа с ним ведется на компьютере.

Гипертекст представляет собой обычный текст с картинками, в котором выделены, чаще всего цветом, отдельные знаки, слова, фразы или картинки, называемые ***полями***. С каждым полем в гипертексте связано определенное действие; чаще всего переход в другое место того же гипертекста или в другой гипертекст. (С некоторыми полями может быть связано воспроизведение фрагмента звукозаписи или видеозаписи либо запуск на выполнение определенной программы.)

При просмотре гипертекста на компьютере достаточно подвести курсор к выделенному месту и нажать всего одну кнопку, и гипертекстовая система мгновенно перенесет вас куда надо, т. е. покажет гипертекст, с которым связано данное поле. При этом новый гипертекст может быть прочитан как с компьютера, на котором вы работаете, так и с любого другого компьютера в Интернете.

Как указать, какой новый гипертекст нужно показать человеку, если он выбрал данное поле? Как задать местоположение нового гипертекста на нашей планете? Подобно любой информации на компьютере, гипертексты хранятся в файлах. Предположим, что

— некоторый гипертекст записан в файле myhypertext.htm;

— этот файл лежит в каталоге mydirectory;

— этот каталог находится на компьютере mycomputer.msk.ru.

Тогда одновременное задание типа файла (.htm), имени файла, имени каталога и имени компьютера в Интернете однозначно определяет, где на планете нужно искать данный файл. Эти данные принято записывать так:

http://mycomputer.msk.ru/mydirectory/myhypertext.htm

Такая запись называется Универсальным Адресом в Интернете, или URL. Сокращение ***URL*** означает ***Universal Resource Locator*** — универсальный адрес ресурса. Вместо слова «файл» здесь употреблено слово «ресурс». Дело в том, что URL может

указывать не только на неизменный файл, записанный где-то на диске, но и на какую-то программу, которая генерирует, создает этот файл в момент запроса. Такая программа, например, может в момент запроса генерировать файл с текущим курсом доллара или сводкой погоды.

37.3. Всемирный гипертекст: HTTP, FTP, ...

На подключенном к сети Интернет компьютере можно запустить специальную программу, которая называется ***www-сервер***, и собрать в один или несколько каталогов файлы, разрешенные к просмотру всему миру. После этого программа будет готова по запросу извне от любого компьютера в Интернет переслать любой запрошенный «разрешенный» файл. Весь мир увидит файл myhypertext.htm, только если этого захочет хозяин подключенного к Интернету компьютера mycomputer.msk.ru. Пока на этом компьютере не запущен www-сервер, в ответ на запросы по посылке информации http://mycomputer.msk.ru/mydirectory/myhypertext.htm будут посылаться отказы.

Четыре буквы ***http*** в начале URL расшифровываются как ***hypertext transfer protocol*** — протокол передачи гипертекстов. Это сокращение означает, что запрос на посылку информации должен выполняться программой www-сервер, а сама информация должна передаваться по специальным правилам. Кроме протоколов передачи гипертекстов есть и другие протоколы, например ***ftp*** protocol — ***file transfer protocol*** — протокол передачи файлов. Для передачи вовне файлов в формате ftp необходимо установить на компьютере еще одну программу — ***ftp-сервер***.

На одном компьютере в Интернете могут одновременно работать несколько программ-серверов: каждая из них готова выполнить запрос извне, каждая ждет очередного запроса. Поэтому при посылке пакета от одного компьютера в Интернете другому нужно указывать не только от какого компьютера какому послан запрос, но и от какой программы компьютера-отправителя он исходит и какой программе компьютера-получателя он адресован. Для задания программ используются целые числа. При формировании пакета эти числа добавляются к IP-адресам отправителя и получателя; называются они ***номерами портов***. Для многих программ общеприняты стандартные номера портов. Например, для HTTP-сервера номер порта равен 80, а для FTP-сервера 20 или 21.

37.4. Язык записи гипертекстов HTML

Четкое описание первой версии Всемирного гипертекста появилось в 1990 г. К середине 90-х гг. компьютерное сообщество, а затем и весь мир безоговорочно признали общемировой стандарт записи гипертекстов, стандарт ***HTML*** (***HyperText Markup Language*** — язык разметки гипертекстов). И полные описания этого стандарта, и короткие руководства для начинающих, в том числе и на русском языке, можно найти в Интернете.

Основные элементы HTML для оформления *обычных* текстов мы рассмотрели в § 3. Но самое интересное в HTML — это гиперссылки, т. е. ссылки в одном тексте на другие тексты, расположенные на том же компьютере или на любом другом компьютере в Интернете. Для превращения обычного текста в *гипертекст* в HTML существует специальный тег <A>. В простейшем случае гиперссылка оформляется так:

<A HREF = "URL"> *текст гиперссылки* </A>

Рассмотрим конкретный пример. Предположим, на некоторой странице есть такой фрагмент текста в HTML:

СТЕНОГРАММА ПРЕСС-КОНФЕРЕНЦИИ </p>

Если при просмотре страницы пользователь выберет выделенные подчеркиванием или цветом слова СТЕНОГРАММА ПРЕСС-КОНФЕРЕНЦИИ, то произойдет переход по указанному в гиперссылке URL, т. е. пользователь увидит:

файл	36.html
в директории	/events
на компьютере	president.kremlin.ru

содержащий официальный документ под названием

«Стенограмма пресс-конференции Президента Российской Федерации В. В. Путина и Президента Соединенных Штатов Америки Б. Клинтона (4 июня 2000 года, г. Москва)»

сделанный доступным всему миру администрацией Президента России.

<A HREF = "http://mycomputer.msk.ru/mydirectory/myhypertext.htm"> мой файл </A>

При отображении этой страницы на экране слова «мой файл» будут выделены цветом и/или подчеркиванием. Если пользователь выберет эту ссылку, произойдет переход по указанному URL.

38 Опасности применения компьютеров

Мы много говорили о достоинствах компьютеров и об их роли в жизни общества. Однако, как и любое другое изобретение человека, компьютер может принести не только пользу, но и вред. Представление о том, когда компьютеры использовать нецелесообразно, каковы основные ошибки в их применениях, является важной частью компьютерной грамотности. Поэтому мы кратко перечислим несколько таких случаев.

1. Превращение компьютера из средства в цель. Применение компьютера само по себе отнюдь не служит признаком технического прогресса. Скорее наоборот — прогресс чаще оказывается связан не с усовершенствованием существующей, а с переходом на новую технологию. Например, переход на точное литье упраздняет чистовую механическую обработку деталей и делает ненужной компьютер, управляющий этой обработкой. Стремление «внедрить компьютер» может воспрепятствовать такому переходу и тем самым затормозить научно-технический прогресс.

Аналогично отмена дополнительной платы за междугородные телефонные разговоры сделает ненужным компьютер, вычисляющий их стоимость в зависимости от длительности разговора и расстояния между городами. Строительство туннелей и эстакад приведет к упразднению светофоров и регулирования движения с помощью компьютера. При переходе к новым принципам оплаты труда, налогового обложения и социального обеспечения станет ненужным расчет зарплаты на компьютере и т. п.

2. Ошибки в алгоритмах. Компьютер лишь выполняет алгоритмы. Эти алгоритмы могут быть составлены с ошибками или на основе неверных представлений о действительности. Например, одна из первых компьютерных систем противовоздушной обороны США (60-е гг.) в первое же дежурство подняла тревогу, приняв восходящую из-за горизонта Луну за вражескую ракету, поскольку этот «объект» приближался к территории США и не подавал сигналов, что он «свой».

3. Неверные исходные данные. Результат работы компьютера зависит не только от алгоритма, но и от обрабатываемой ин-

формации. В исходных данных ошибки не менее опасны, чем в алгоритмах. Так, несколько лет назад в Антарктиде разбился самолет с туристами на борту, поскольку в управляющий полетом компьютер были помещены неверные координаты аэропорта взлета и компьютер ошибочно рассчитал высоту полета над горами.

4. Компьютеры не всемогущи. Далеко не всякая задача обработки информации может быть решена с помощью компьютера. Существуют задачи, алгоритмы решения которых в настоящее время неизвестны. Например, до сих пор не существует приемлемых алгоритмов, которые позволили бы отличить на фотографии кошку от собаки или грамотно перевести художественное произведение с одного языка на другой. Бывает и такое, что алгоритм известен, но выполнить его нельзя, поскольку даже самым быстродействующим компьютерам для его выполнения понадобятся миллионы лет (пример такой задачи — безошибочная игра в шахматы). Поэтому в корне неверно представление о том, что если человек не знает решения задачи, то ее надо «заложить в компьютер» и компьютер даст ответ.

5. Недооценка социальных последствий компьютеризации. Наконец, и это самое важное, использование компьютера меняет жизнь людей. Поэтому вопрос о новых применениях компьютера прежде всего должен рассматриваться с точки зрения социальных последствий, а не с позиции «могут это компьютеры» или «не могут», выгодно это или не выгодно. Многие этапы информатизации общества имеют трудно предсказуемые социальные последствия. Внедрение заводов-автоматов требует перевода значительной части работающих из производственной сферы в сферу обслуживания. Если работа в сфере обслуживания считается в обществе менее престижной, такой перевод может вызвать социальную напряженность. Организация работы на дому позволяет увеличить количество свободного времени, но разрушает сферу общения с сослуживцами. Распространение компьютерных игр приводит к тому, что дети быстрее развиваются, но меньше бывают на воздухе и меньше общаются друг с другом. Во многих случаях компьютер просто не следует внедрять. Например, в вопросах, связанных с принятием моральных и этических решений при воспитании детей, формулировании целей социального развития общества, установлении виновности обвиняемых в преступлении.

Оглавление