ЗАОЧНАЯ ШКОЛА СОВРЕМЕННОГО ПРОГРАММИРОВАНИЯ

Обучение искусству программирования требует изучения таких дисциплин, как логика, дискретная математика, прикладная теория алгоритмов и др., которым в школьном курсе уделяется совсем немного времени, а во многих школах их не изучают вовсе. Школа современного программирования ставит своей целью восполнить этот пробел, а также приобщить школьников, интересующихся профессией программиста, к решению содержательных задач по программированию. Предполагается, что в школе будут обучаться как те ребята, которые владеют теми или иными языками программирования, так и те, которые не имеют о них пока никакого представления.

В соответствии с этим, в школу принимаются ребята 5-11 классов, а предлагаемые задачи имеют два уровня (***Уровень 1*** не требует умения программировать). Каждый обучающийся может выбрать тот уровень, который окажется ему по силам.

Обучение ведется на модульной основе, поэтому прием в школу осуществляется непрерывно. Учащиеся получают материалы всех занятий текущего календарного года.

Все подписчики журнала найдут условия задач в очередном номере журнала, начиная с первого. Тем, кто не является подписчиком, задания будут высылаться по электронной или обычной почте.

Решения принимаются на дискетах и в письменном виде (по адресу: 191025, Санкт-Петербург, Марата 25, Заочная школа современного программирования, телефон для справок (812) 164-13-55) и по электронной почте (school@aec.neva.ru). Обращаем внимание, что участниками школы могут быть как отдельные школьники, так и коллективы (классы, кружки и пр.).

УСЛОВИЯ, КОТОРЫМ ДОЛЖНЫ УДОВЛЕТВОРЯТЬ ПРИСЫЛАЕМЫЕ РЕШЕНИЯ.

Программы (***Уровень 2***) проверяются автоматической системой тестирования, поэтому необходимо точно следовать указанным в условиях форматам ввода/вывода, а также указанным ниже правилам.

Все программы должны быть скомпилированы под DOS и должны использовать стандартный ввод/вывод. Для Паскаля это процедуры read/readln и write/writeln, для Си - функции scanf и printf, для Си++ - объекты cin и cout, для Бейсика - операторы INPUT и PRINT. Нельзя использовать модули или библиотеки, которые перенаправляют стандартный ввод/вывод (например, для Паскаля - модуль CRT). При одинаковых входных данных программа должна выдавать одинаковый результат.

Названия исполнимых файлов с программой:
<первые три буквы фамилии>_<задача>.exe

Например, pet_2.exe, sid_3.exe.

Все тексты должны быть набраны в формате ASCII.

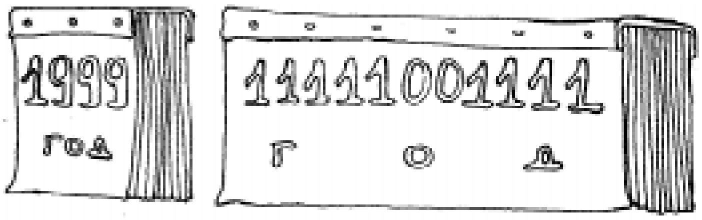
Тексты программ будут читаться проверяющими, поэтому следует использовать поясняющие комментарии и уделять внимание структуре программы, а также названиям идентификаторов.

Примечание: те, кто не может прислать решения в электронном виде, присылают подробно и аккуратно оформленные решения на бумаге.

ЗАНЯТИЕ 2. НАБОРЫ ИЗ НУЛЕЙ И ЕДИНИЦ

Когда мы начинаем заниматься программированием, мы узнаем о двоичной системе счисления... А вы еще ничего не узнали? Ну, тогда несколько слов для тех, кто еще не узнал.

В двоичной системе счисления, в отличие от привычной для нас десятичной, всего две цифры - 0 и 1. Счет идет так: 0, 1, потом сразу 10. Потом 11, это естественно, ведь всегда 10+1=11. Потом 100, если подумать, то тоже естественно, когда к единице в первом разряде добавляешь единицу, то получится 10, 0 пишем, а единицу переносим в старший разряд, где тоже получается 10. Потом (уже проще) 101, 110, 111, потом (уже можем догадаться) 1000, и так далее.



Числа при записи в этой системе получаются очень длинными, например, 1999 выглядит так: 11111001111. Почему? Дело в том, что в этой системе числа 10, 100, 1000 и т.д. - это степени двойки: 100 = 4, 1000 = 8, 10000 = 16. Полученная двоичная запись выражает равенство
1999 = 1024 + 512 + 256 + 128 + 64 +
+ 8 + 4 + 2 + 1.
Нули в записи соответствуют отсутствующим слагаемым 16 и 32.

В этой системе очень просто складывать числа, а уж прибавить к числу единицу... - ну, просто ничего не стоит. Нужно просматривать запись с конца и, если встречается единица, заменить ее нулем, а если нуль, то заменить его единицей и остановить просмотр. Так что, 11111001111 + 1 = = 11111010000 (так должна выглядеть запись числа 2000. Проверьте).

Числа, записанные в двоичной системе, удобно умножать и делить на 2, ведь два - это двоичное десять. Значит, для умножения нужно просто приписать в конце нуль, а для деления - стереть последний разряд, который должен быть нулем. Если же в конце стоит единица, то при делении получится остаток. Так что, мы сразу видим, что число 11111010000 можно разделить на два без остатка четыре раза.

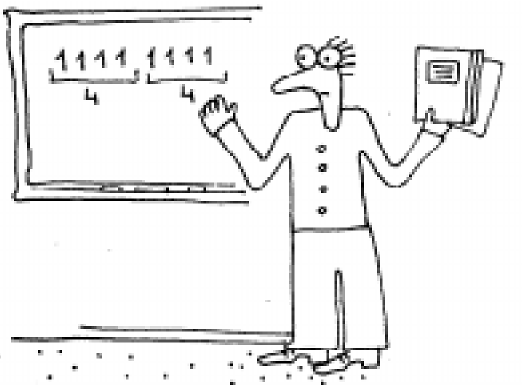
Наборы из нулей и единиц важны из-за того, что память компьютера состоит из мельчайших элементов (битов), каждый из которых может хранить одну двоичную цифру. Очень скромная память в 1 мегабайт* содержит больше 8 миллионов таких битов.

Представьте:
0100101000100101000001011111110101101011001111100010100001110001,
как в таком ужасе разбираться?

*Частичка "кило" (а иногда и "мега") встречаются и в обычной жизни, хотя для многих из нас работа с компьютером - это уже обычная жизнь. Вспомним километр, килокалорию и др. В обычном использовании, вы помните, кило обозначает ровно тысячу. Некоторые предлагали писать по-разному обычные и компьютерные кило. Но все решили, что всегда и так понятно, что имеется в виду. А вы понимаете? Если подержанный автомобиль продается за $2K, какая сумма имеется в виду?

Обозначение	Название	Смысл	Точное значение	Степень двойки
K	Кило	Тысяча	1024	10
M	Мега	Миллион	1048576	20
Г	Гига	Миллиард	1073741824	30
T	Тера	Триллион	1099511627776	40

Обычно строку из битов разбивают на четверки или иначе тетрады (не путать с тетрадями, хотя происхождение этих двух слов одно и то же):
0100 1010 0010 0101 0000 0101 1111 1101 0110 1011 0011 1110 0010 1000 0111 0001.



В свою очередь, тетрада может трактоваться как цифра в системе счисления с основанием 16 - *шестнадцатеричной* системе счисления. Приняты следующие обозначения для цифр в этой системе: от 0 до 9 обычные цифры, а дальше первые шесть букв латинского алфавита: A, B, C, D, E, F. Легко выписать соответствие между тетрадами, цифрами и числами, которые в них закодированы (табл. 1).

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

Таблица 1.

Руководствуясь этой таблицей, мы можем нашу строку переписать так (проверьте): 4 A 2 5 0 5 F D 4 B 3 E 2 8 7 1.

При работе с памятью компьютера удобны более крупные единицы. В качестве самой маленькой единицы памяти принимается восьмерка битов, которую именуют байтом. Байт состоит из двух тетрад, - старшей и младшей, так что содержимое байта записывается двумя шестнадцатеричными цифрами, как мы привыкли:
4A 25 05 FD 4B 3E 28 71.

Для решения в машине многочисленных задач хранения и переработки текстовой информации приходится вводить специальные способы кодирования букв и знаков. В языках, использующих латиницу, постепенно завоевал всемирное признание код **ASCII** - **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange, первоначально разработанный и стандартизованный в США. В этом коде твердо фиксированы первые 128 возможностей, вторая часть может варьироваться. Вот эта вторая часть и используется для кириллицы, причем, к сожалению, общепринятой кодировки нет. Наиболее популярна у специалистов по программированию при работе в MS DOS альтернативная кодировка, которая задается приводимой здесь таблицей* (табл. 2). Строки соответствуют старшему полубайту, столбцы - младшему. (В системе Windows используется другая кодировка).

Важность наборов из нулей и единиц (или просто, 0-1 наборов) связана с тем, что и в компьютерах и в других приложениях они появляются в самых разно-

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	/
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
8	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
9	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
A	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
B				│	┤	╡	╢	╖	╕	╣	║	╗	╝	╜	╛	┐
C	└	┴	┬	├	─	┼	╞	╟	╔	╚	╩	╦	╠	═	╬	╧
D	╨	╤	╥	╙	╘	╒	╓	╫	╪	┘	┌	█	▄	▌	▐	▀
E	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я

Таблица 2.

* В таблицу не вписаны две первые строки,. в которых содержатся управляющие коды, и последняя строка, в которой символы слишком сложные.

образных математических моделях, и часто оказывается полезно знать и совместно использовать различные варианты их трактовки. Попробуем некоторые из них перечислить.

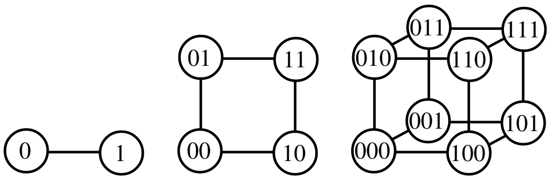


Рисунок 1. *Вершины единичного куба.*

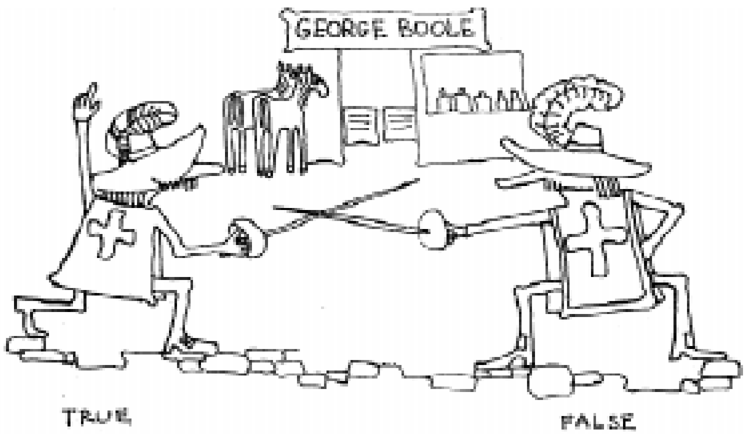
А. Прежде всего, мы можем считать, что набор из нулей и единиц - это... точка многомерного евклидова пространства, причем он определяет *одну из вершин куба*, построенного на координатных ортах (рисунок 1). Вы знакомы с двумерным, а некоторые и с трехмерным евклидовым пространством. Но мы можем представить себе и более сложные пространства, а единичные кубы нам в этом помогут. Когда наш набор состоит из m чисел, мы говорим об m-мерном пространстве, а сам набор называем вектором в этом пространстве.

В. Такой набор может рассматриваться и как способ задания *подмножества* множества целых чисел от 1 до m (обозначение $1{:}m$). Именно, если i-ый элемент набора равен 1, то число i принадлежит множеству, в противном случае, не принадлежит. Так определенный набор называется *характеристическим вектором* подмножества.

С помощью характеристических векторов удобно описывать операции над множествами: например, характеристический вектор пересечения двух множеств имеет i-ой компонентой 1, если i-ые компоненты всех характеристических векторов пересекаемых множеств равны 1.

<u>*Упражнение.*</u> Сформулируйте правила образования характеристических векторов объединения, разности и симметрической разности множеств*.

С. Очень близка к этой трактовке логическая интерпретация векторов из нулей и единиц. В XIX веке английский математик Дж.Буль (George Boole, 1815-1864) предложил для математического моделирования понятий формальной логики использовать специальные переменные (называемые сейчас *логическими* или *булевыми*). Эти переменные могут принимать только два значения: **TRUE** и **FALSE** (**ИСТИНА** и **ЛОЖЬ**).



Над логическими величинами можно выполнять специальные *логические операции*.

Одноместная операция (то есть, имеющая один операнд) **NOT** (**НЕТ**) описывает *отрицание*, - она вырабатывает значение **TRUE** при значении аргумента **FALSE**, и наоборот (в математической логике эту операцию обозначают $\neg$).

Операция *логического умножения* **AND** (**И**), имеющая два операнда, вырабатывает значение **TRUE** только тогда, когда истинны оба операнда (эта операция называется также *конъюнкцией* и обо-

* Симметрической разностью множеств А и В называется множество элементов, которые входят в одно из этих множеств, но не входят в другое.

значается $\wedge$ или **&**. Это можно сказать и по-другому: операция вырабатывает логическое значение “оба операнда истинны”.

Операция *логического сложения* **OR** (**ИЛИ**) вырабатывает значение **TRUE** только тогда, когда истинен хотя бы один из операндов (эта операция называется *дизъюнкцией* (disjunction) и обозначается $\vee$ или |). Опять-таки, можно сформулировать это так: операция вырабатывает значение “хотя бы один из операндов истинен”.

Операция **EQU** (EQUivalence — эквивалентность) или $\equiv$ вырабатывает значение “операнды равны”, она называется *тождественностью.* Наряду с ней, встречается операция **XOR** (eXclusive OR — Исключающее ИЛИ) или $\not\equiv$, вырабатывающая ее отрицание, то есть значение “истинен ровно один из ее операндов”.

<u>*Упражнения.*</u>

1. Докажите, что (a **XOR** b) **XOR** $b = a$ (это свойство операции XOR очень удобно в компьютерной графике).
2. Как мы знаем, точку на окружности можно задавать числом от 0 до 2π - углом поворота от начальной оси. Отрезок на окружности определяется упорядоченной парой (a,b). Докажите, что
$x \in (a,b) \equiv ((a<b) \equiv ((a<x) \equiv (x<b)))$.

Все эти операции легко формулируются в терминах нулей и единиц (таблица 3). Их можно перенести на векторы, потребовав, чтобы операции над векторами выполнялись “покомпонентно”, то есть независимо над соответствующими элементами - компонентами векторов-операндов.

В таблице 4 пример, в котором все названные логические операции выполняются над двумя векторами.

Сравнение двух последних строчек дает пример действия операции отрицания **NOT**.

Иногда удается получать интересные вычислительные эффекты сочетанием арифметических и логических операций.

В качестве примера рассмотрим способ поиска единиц в векторе. Каждый раз будет разыскиваться самый младший разряд с единицей. Вектор, в котором идет поиск, рассматривается одновременно и как набор битов и как двоичное представление целого числа. Первоначально вычтем из этого числа единицу. Это действие “разменяет” младшую единицу в последовательность единиц нижестоящих разрядов. Так, если мы имели число
a = 0001011100011000,
после вычитания единицы мы получим
b = 0001011100010111.
Теперь действие **XOR** выделит разряды, в которых эти два набора отличаются:
c = 0000000000001111, после чего достаточно логически умножить этот набор на a, чтобы получить окончательный результат.

D. Мы уже говорили об интерпретации вектора из нулей и единиц как состояния памяти вычислительной машины.

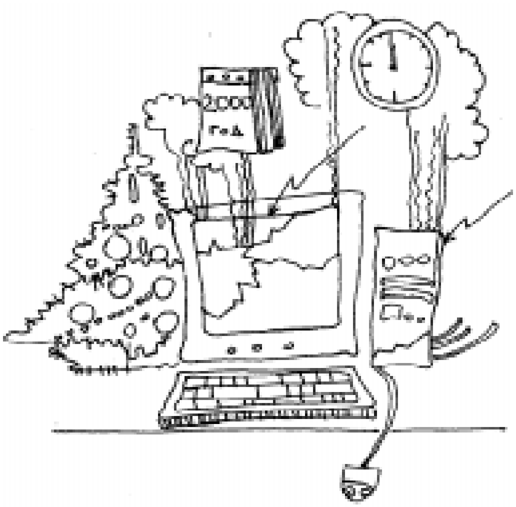
x	y	$x \wedge y$	$x \vee y$	$x \equiv y$	$x \not\equiv y$
0	0	0	0	1	0
0	1	0	1	0	1
1	0	0	1	0	1
1	1	1	1	1	0

Таблица 3.

x	00011101010111001000111011101
y	11101010111010110101110101010
$x \wedge y$	00001000010010000000110001000
$x \vee y$	11111111111111111110111111111
$x \equiv y$	00001000010010000010110001000
$\neg(x \equiv y)$	11110111101101111101001110111

Таблица 4.

Для такой трактовки существенно, что, в зависимости от обстоятельств, содержимое памяти может трактоваться по-разному. Например, в операционной системе MS DOS для запоминания даты создания информационного объекта (файла) используется два байта - 16 битов. Семь старших битов - это год (считая нулевым годом 1980-ый), следующие четыре бита - месяц (с некоторым запасом), последние пять - день месяца. В частности, пара байтов 06 27 интерпретируется и как число 1575, и как дата 7 февраля 1983 г. (а почему не января?).



E. Последовательность нулей и единиц может еще рассматриваться и как *сообщение*, *передаваемое по каналу связи* (передаются импульсы, каждый из которых принимает одно из двух значений), и как запись результатов экспериментов, каждый из которых может кончиться успехом (1) или неудачей (0).

F. В некоторых случаях полезно представлять себе эту последовательность как “монотонный” путь на прямоугольной решетке, в котором, например, нули соответствуют шагам направо, а единицы — шагам наверх. На рис. 2 представлен путь (1,0,0,0,0,0,1,1,0).

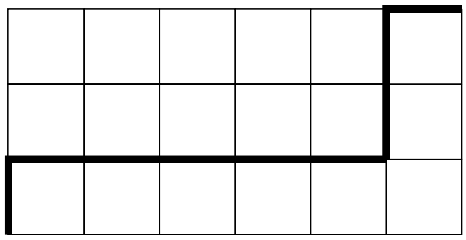
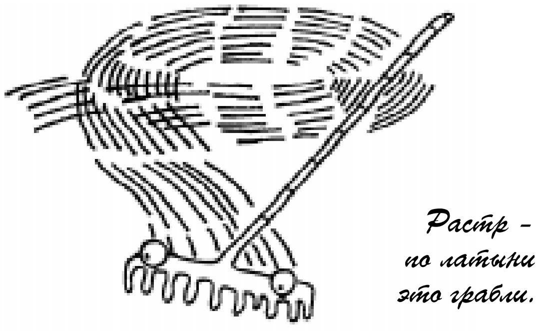


Рисунок 2.

G. Еще одна очень важная трактовка набора из нулей и единиц — это кодировка геометрического изображения. Двухцветная картинка (будем по традиции говорить о черно-белой картинке - черный рисунок на белом фоне) может трактоваться как *растр* - совокупность отдельных точек, расставленных на прямоугольной решетке.



Сопоставляя черным точкам единицы, а белым нули, мы и закодируем нашу картинку в виде набора нулей и единиц.

<u>*Пример.*</u> Рассмотрим решетку 16×16 и картинку на ней, изображенные на рисунке 3. Кодируя столбцы числами в диапазоне $0{:}2^{16}-1$, получаем вектор, который удобнее представить в шестнадцатеричной системе:
0000, 0000, 1800, 2C00, 2604, 7208, 4208, 4605, 2B83, 3C7C, 0C08, 0030, 0020, 0010, 0000, 0000.

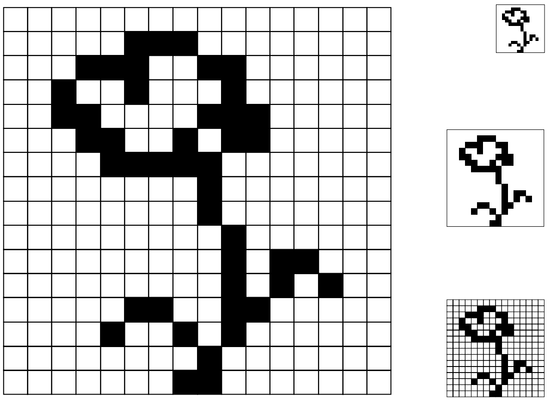


Рисунок 3. *Битовый скромный цветочек.*

Упражнение. Проверьте правильность кодировки и установите, какой край картинки выбран для младших разрядов.

Нарисуйте сами и закодируйте какой-либо символ 8×8.

Эта возможность рисования картинок точками широко используется в вычислительной технике и в обработке изображений. Например, экран дисплея рассматривается как такой растр, в котором (фиксируем конкретную ЭВМ, очень старую) 350 строк по 640 точек в каждой строке, то есть в общей сложности 224000 точек. Для сохранения такого набора нужно (по 8 точек в одном байте) 28000 байтов.

Символы, изображаемые на дисплее или на печатающем устройстве, также имеют растровое представление. При этом растр, в зависимости от конкретного устройства, трактуется как последовательность строк (дисплей и символы на нем) или как последовательность столбцов (матричный принтер).

Н. Многократное использование таких элементарных конструкций может привести к довольно сложным структурам данных (естественно, все, что есть в компьютерах, состоит из нулей и единиц). Рассмотрим в качестве примера структуру данных в видеопамяти EGA при использовании цветового графического режима (и приводимые здесь параметры дисплея и адаптер EGA — Enhanced Graphical Adapter — безнадежно устарели и не используются. Но этот адаптер удобнее для иллюстрации, чем более простые современные). Как было сказано, растр дисплея составляет 224 000 точек. В цветовом режиме каждой из этих точек можно сопоставить один из 16 цветов, что, очевидно, требует 4 бита на каждую точку.

Эти биты в системе EGA было принято располагать не все вместе, а в отдельных областях памяти по 224 000 битов каждый (на самом деле, конечно, по 256 килобитов = 32 килобайта). Эти области традиционно называются *цветовыми плоскостями*. Имеется возможность записи информации одновременно во все цветовые плоскости. Однако, поскольку адресуемой единицей информации в компьютере является байт, запись естественно вести сразу в восемь смежных битов.

Чтобы все-таки иметь возможность выбирать в байте те биты, в которые требуется заносить информацию, предусмотрена система “маски”. Именно, специальной командой устанавливается байт, который определяет “картинку”, которая заносится в видеопамять. Здесь мы видим еще одно использование 0-1 вектора.

Система VGA, в которой число допустимых цветов значительно больше, аналогична, но в ней на задание каждого цвета отводится по 2 байта.

I. Поговорим теперь про отрицательные числа. Представление целых чисел любого знака требует новых решений. Из многих мыслимых вариантов устойчиво закрепились два, - кодирование со смещением и дополнительный до двух код.

При *кодировании со смещением* к каждому числу r прибавляется константа, выбранная так, чтобы при любом r сумма $r+D$ была неотрицательной, и эта сумма кодируется вместо r. Остается решить, какой должна быть эта константа, но на этот вопрос ответить несложно. Именно, если для представления чисел выбрали k двоичных разрядов, то можно закодировать 2^k различных чисел. Будет естественно поделить эти возможности поровну между положительными и отрицательными числами. Правда, еще есть нуль. Припишем его к положительным числам, так удобнее. Тогда константа смещения равна $D = 2^{k-1}$. Например, если k = 5, то $D = 2^4 = 16$ и code(0) = 16, code(11) = 27, code(-7) = 9, а code(17) и code(-19) не определены, так как эти числа выходят за границы интервала определения кода – 16:15.

Дополнительный код оперирует с остатками от деления чисел на 2^k, при этом остаток r, лежащий в интервале $[\,2^{k-1};2^k-1\,]$, считается отрицательным числом $r\text{-}2^k$. Таким образом, для тех же примеров code(0) = 0, code(11) = 11, code(-7) = 32-7 = 25. В основной части арифметических расчетов используется дополнительный код, так как расчеты с этой записью проводятся легче и естественнее, код со смещением применяется только в специальных случаях.

Упражнения.

1. Чем отличаются правила сложения чисел, заданных в обеих кодировках, и правила их сравнения?
2. Определите правила перехода от одной кодировки к другой.

J. Еще один пример системы, построенной над двоичными системами, дают *штриховые коды* (barcodes). Их используют во многих практических информационных системах, - в магазинах, в почтовой службе, и других местах, где требуется быстро и просто считывать в компьютер небольшую числовую или текстовую информацию. Мы встречаем их на упаковках товаров, на книгах. Имеется много разновидностей этих кодов, и пример одной из них, так называемого *кода 3 из 9* представлен на рис. 4.

Каждый символ кодируется 9 полосками, поочередно черными и белыми, в

Рисунок 4.

конце добавляется еще одна белая полоска. Три полоски из основных девяти имеют увеличенную ширину. Так можно закодировать $9\times8\times7/6 = 84$ разных символов, что позволяет закодировать весь латинский алфавит (хотя на самом деле кодируются только прописные буквы), цифры и некоторые знаки. Сопоставляя каждой широкой полоске 1, а узкой полоске 0, получаем 10-битовые последовательности, в которых последний бит всегда равен 0, так что существенными для информации оказываются 9 битов. Посмотрим, как трактуются эти биты.

Каждой кодовой комбинации соответствует трехэлементное подмножество, которое задается вектором из трех единиц и 6 нулей. Для цифр и нулей выбраны такие коды, в которых одна широкая белая полоска и две черных. Удобно определять отдельно векторы черных и белых полосок. Для цифр в качестве вектора белых полосок берется вектор [0, 1, 0, 0], а в качестве векторов черных полосок для цифр 1, 2, 3,..., 9, 0 берутся подряд векторы [1, 0, 0, 0], [0, 1, 0, 0], [1, 1, 0, 0] и т.д. кроме тех, где больше двух единиц. Пятая компонента вектора выбирается так, чтобы дополнить число единиц до 2. Таким образом, для 3 мы имеем [1, 1, 0, 0, 0], а для 7 — [0, 0, 0, 1, 1].

Буквы A - J кодируются как цифры, но с вектором белых полосок [0, 0, 1, 0], буквы K - T с вектором [0, 0, 0, 1], буквы U - Z, тире, запятая, пробел и звездочка - с вектором [1, 0, 0, 0].

Теперь можно проверить пример на рисунке 4. Составим таблицу (табл. 5), в

Знак	**Ч** Б **Ч** Б **Ч** Б **Ч** Б **Ч**	код черн.	код бел.	группа	номер
*	**У** Ш **У** У **Ш** У **Ш** У **У**	00110	1000	U	5
B	**У** У **Ш** У **У** Ш **У** У **Ш**	01001	0010	A	2
A	**Ш** У **У** У **У** Ш **У** У **Ш**	10001	0010	A	1
R	**Ш** У **У** У **У** У **Ш** У **Ш**	10010	0001	K	8
C	**Ш** У **Ш** У **У** Ш **У** У **У**	11000	0010	A	3
O	**Ш** У **У** У **Ш** У **У** Ш **У**	10100	0001	K	5
D	**У** У **У** У **Ш** Ш **У** У **Ш**	00101	0010	A	4
E	**Ш** У **У** У **Ш** Ш **У** У **У**	10100	0010	A	5
3	**Ш** У **Ш** Ш **У** У **У** У **У**	11000	0100	1	3
9	**У** У **Ш** Ш **У** У **Ш** У **У**	01010	0100	1	9

Таблица 5.

которой первый столбец заполнен символами, далее следует 9 знаков, описывающих ширину колонок (У - узкая, Ш - широкая), затем по этим знакам сформированы коды черных и белых полосок, и, наконец, эти коды переведены в условные обозначения группы и номера в группе. В качестве представителя в каждой группе выбран символ с кодом 1. Напомним, что пропуск в кодировке чисел 7 и 11, имеющих слишком много единиц, сдвинул нумерацию, и это повлияло на представление кода черных полосок у символов R и 9.

В связи с развитием так называемых multimedia - всевозможных периферийных видео- и аудиоустройств, появились многочисленные новые форматы, в частности, для звуковых файлов и кинофайлов (например, формат AVI). Имеется много специальных форматов графического вывода (например, метафайлы Windows), которых просто не перечесть. Вы можете найти богатую информацию в многочисленных сейчас справочниках по графическим форматам. Об одной группе форматов нужно упомянуть особо, - о форматах сжатия текстов.