Представление символов в памяти ЭВМ

Общий принцип кодирования символов очень прост – каждому символу присваивается свой уникальный числовой идентификатор при помощи которого можно однозначно определить символ и отобразить его на экране. Т.е. в файлах хранятся не сами изображения символов, а их числовые коды, которые можно сопоставить с реальными изображениями символов, хранящихся в системе в единственном виде.

Но так как данные хранятся в памяти в виде непрерывной последовательности нулей и единиц и ничем друг от друга не отделены, для работы с кодами символов как и для работы с целыми и вещественными числами нужно знать сколько бит занимает код символа. Из-за этого возникает множество проблем.

Самая первая общепринятая кодировка была создана в США и называлась таблица ASCII (American Standard Code for Information Interchange). Каждый код символа в этой кодировке занимал 8 бит, первые 32 символа использовались для непечатных символов при помощи которых осуществлялось управление периферийными устройствами, остальные символы таблицы – английские буквы в большом и малом регистре, цифры, знаки препинания, арифметические символы и некоторые часто использующиеся в США символы (скобки, %, $, & и тд), всего 128 символов. Из-за того, что кодировка разрабатывалась как американский стандарт, она не была рассчитана на использование в других странах. Поэтому в кодировке отсутствовали символы других алфавитов, даже латинских символов, используемых в Западной Европе.

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Таблица 1 – таблица ASCII

После того как компьютеры стали применяться в остальном мире возникла необходимость кодировать и другие символы. Для этого было принято решение использовать старший бит восьмибитной кодировки, который до этого использовался в служебных целях (т.к. все коды в таблице ASCII были в диапазоне от 0 до 127 старший бит был не нужен для представления кодов символов). Первая таблица, использующая все восемь бит для представления кодов символов была “расширенная таблица ASCII” (extended ASCII). Она включала в себя варианты латинских символов, использующихся в языках Западной Европы, псевдографические символы и некоторые дополнительные символы.

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	═	237	ED	ø
142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
144	90	É	176	B0	░	208	D0	╨	240	F0	≡
145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
155	9B	¢	187	BB	╗	219	DB	█	251	FB	√
156	9C	£	188	BC	╝	220	DC	▄	252	FC	ⁿ
157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	

Таблица 2 – расширенная часть таблицы ASCII

В других странах разрабатывались свои кодировки, позволяющие представлять специфические национальные символы. При этом у всех вариантов таблицы ASCII первые 128 символов одинаковые, т.е. совпадают с первоначальной таблицей ASCII.

Недостатки кодирования символов при помощи таблиц ASCII очевидны – из-за ограничения размера кода символа в восемь бит далеко не всегда можно включить в кодировку все нужные символы и приходится выбирать исходя из определенных потребностей, что в свою очередь порождает множества вариантов таблиц ASCII даже в рамках одной страны (самых распространённых кодировок, поддерживающих русский язык, было три, фактически их было больше). При этом возникает проблема определения кодировки для принимаемых данных. Если ПО не в состоянии автоматически определить кодировку текста, делать это приходилось вручную.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
80	2500 ─	2502 │	250C ┌	2510 ┐	2514 └	2518 ┘	251C ├	2524 ┤	252C ┬	2534 ┴	253C ┼	2580 ▀	2584 ▄	2588 █	258C ▌	2590 ▐
90	2591 ░	2592 ▒	2593 ▓	2320 ⌠	25A0 ■	2219 ∙	221A √	2248 ≈	2264 ≤	2265 ≥	A0	2321 ⌡	B0 °	B2 ²	B7 ·	F7 ÷
A0	2550 ═	2551 ║	2552 ╒	451 ё	2553 ╓	2554 ╔	2555 ╕	2556 ╖	2557 ╗	2558 ╘	2559 ╙	255A ╚	255B ╛	255C ╜	255D ╝	255E ╞
B0	255F ╟	2560 ╠	2561 ╡	401 Ё	2562 ╢	2563 ╣	2564 ╤	2565 ╥	2566 ╦	2567 ╧	2568 ╨	2569 ╩	256A ╪	256B ╫	256C ╬	A9 ©
C0	44E ю	430 а	431 б	446 ц	434 д	435 е	444 ф	433 г	445 х	438 и	439 й	43A к	43B л	43C м	43D н	43E о
D0	43F п	44F я	440 р	441 с	442 т	443 у	436 ж	432 в	44C ь	44B ы	437 з	448 ш	44D э	449 щ	447 ч	44A ъ
E0	42E Ю	410 А	411 Б	426 Ц	414 Д	415 Е	424 Ф	413 Г	425 Х	418 И	419 Й	41A К	41B Л	41C М	41D Н	41E О
F0	41F П	42F Я	420 Р	421 С	422 Т	423 У	416 Ж	412 В	42C Ь	42B Ы	417 З	428 Ш	42D Э	429 Щ	427 Ч	42A Ъ

Таблица 3 – кодировка КОИ-8R

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
80	402 Ђ	403 Ѓ	201A ‚	453 ѓ	201E „	2026 …	2020 †	2021 ‡	20AC €	2030 ‰	409 Љ	2039 ‹	40A Њ	40C Ќ	40B Ћ	40F Џ
90	452 ђ	2018 ‘	2019 ’	201C “	201D ”	2022 •	2013 –	2014 —	□	2122 ™	459 љ	203A ›	45A њ	45C ќ	45B ћ	45F џ
A0	A0	40E Ў	45E ў	408 Ј	A4 ¤	490 Ґ	A6 ¦	A7 §	401 Ё	A9 ©	404 Є	AB «	AC ¬	AD -	AE ®	407 Ї
B0	B0 °	B1 ±	406 І	456 і	491 ґ	B5 µ	B6 ¶	B7 ·	451 ё	2116 №	454 є	BB »	458 ј	405 Ѕ	455 ѕ	457 ї
C0	410 А	411 Б	412 В	413 Г	414 Д	415 Е	416 Ж	417 З	418 И	419 Й	41A К	41B Л	41C М	41D Н	41E О	41F П
D0	420 Р	421 С	422 Т	423 У	424 Ф	425 Х	426 Ц	427 Ч	428 Ш	429 Щ	42A Ъ	42B Ы	42C Ь	42D Э	42E Ю	42F Я
E0	430 а	431 б	432 в	433 г	434 д	435 е	436 ж	437 з	438 и	439 й	43A к	43B л	43C м	43D н	43E о	43F п
F0	440 р	441 с	442 т	443 у	444 ф	445 х	446 ц	447 ч	448 ш	449 щ	44A ъ	44B ы	44C ь	44D э	44E ю	44F я

Таблица 4 – кодировка CP1251

Для решения проблем связанных с представлением символов в памяти ЭВМ в 1991 году некоммерческой организацией “Консорциум Юникода” был предложен стандарт, обеспечивающий представление всех письменностей мира и специальных символов (стандарт UNICODE).

Некоторые кодировки стандарта позволяют кодировать 2 147 483 648 символов, для совместимости с остальными кодировками было принято использовать 1 112 064 символа. Но и этого более чем достаточно, в данный момент используется только около 110 000 кодовых позиций.

Как и в таблицах ASCII каждому символу в UNICODE соответствует уникальный числовой код, для обозначения которого используется запись вида U+xxxx или U+xxxxx или U+xxxxxx, где x – шестнадцатеричная цифра. Всё кодовое пространство стандарта разбито на 17 плоскостей. Чем больше порядковый номер плоскости тем больше числа, кодирующие символы, входят в эту плоскость.

Плоскость	Название	Диапазон символов
Plane 0	Basic multilingual plane (BMP)	U+0000…U+FFFF
Plane 1	Supplementary multilingual plane (SMP)	U+10000…U+1FFFF
Plane 2	Supplementary ideographic plane (SIP)	U+20000…U+2FFFF
Planes 3-13	Unassigned	U+30000…U+DFFFF
Plane 14	Supplementary special-purpose plane (SSP)	U+E0000…U+EFFFF
Planes 15-16	Supplementary private use area (S PUA A/B)	U+F0000…U+10FFFF

Таблица 5 – плоскости Юникода

При этом символы распределены таким образом, что наиболее употребимые символы располагаются в младших плоскостях, что позволяет экономить память при хранении данных, состоящих из общеупотребимых символов.

Печатные символы в Юникоде делятся на базовые и модифицирующие (например символ ударения), но модифицирующие символы не могут встречаться как самостоятельные символы. Например символ “Й” (U+0419) может быть представлен в виде пары символов - базового символа “И” (U+0418) и модифицирующего символа с кодом U+0306.

Юникод имеет несколько форм представления: UTF-8, UTF-16, UTF-32.

UTF-8 – кодировка с переменным размером кода символа, обеспечивающая наилучшую совместимость со старыми системами, использующими восьмибитные символы. Код символа UTF-8 получается из кода UNICODE следующим образом:

1. Если код символа в таблице Юникод лежит в диапазоне от 0 до $7F_{16}$ (сюда входят символы таблицы ASCII), представление символа в UTF-8 ничем не отличается от представления в таблице Юникод.
2. Если код символа в таблице Юникод лежит в диапазоне от 80_{16} до $7FF_{16}$ (сюда входят кириллица, расширенная латиница, арабский алфавит, армянский, греческий, еврейский, коптский алфавиты, сирийское письмо, некоторые знаки препинания, международный фонетический алфавит, тана, нко), представление символа в UTF-8 составляется по следующем шаблону 110xxxxx 10xxxxxx, где x – двоичный разряд кода символа в Юникод.
3. Если код символа в таблице Юникод лежит в диапазоне от 800_{16} до $FFFF_{16}$ (оставшиеся формы современной письменности, сложные знаки препинания, математические и другие специальные символы), представление символа в UTF-8 составляется по следующем шаблону 1110xxxx 10xxxxxx 10xxxxxx, где x – двоичный разряд кода символа в Юникод.
4. Если код символа в таблице Юникод лежит в диапазоне от 10000_{16} до $1FFFFF_{16}$ (музыкальные символы, редкие китайские иероглифы, вымершие формы письменности), представление символа в UTF-8 составляется по следующем шаблону 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx, где x – двоичный разряд кода символа в Юникод.

Т.е. принципы кодирования символов в UTF-8 следующие:

1. Если код символа в таблице Юникод занимает 1 байт, а старший бит кода равен нулю, в UTF-8 код символа остается без изменений.
2. В остальных случаях старшие биты старшего байта кода символа в UTF-8 несут информацию о количестве байт, которые кодируют символ, в единичной системе счисления. После информации о количестве байт идёт нулевой бит, отделяющий служебные данные от полезных внутри старшего байта. Младшие байты начинаются с последовательности 10 - признак продолжения кода символа, после него идут биты кода в таблице Юникод.

Например символ “Й” с кодом в таблице Юникод U+0419 представляется в UTF-8 следующим образом:

1. Переводим код символа в двоичную систему счисления: $0419_{16} = 10000011001_2$ (незначащие нули были убраны)
2. Код символа по таблице Юникод U+0419 и входит в диапазон от 80_{16} до $7FF_{16}$, что соответствует переводу кода в UTF-8 по шаблону 110xxxxx 10xxxxxx.
3. Подставляя в шаблон значащие разряды из кода Юникод получаем следующую последовательность нулей и единиц:
   10000011001 → 11010000 10011001 → $D099_{16}$ (слева показан код символа в таблице Юникод в двоичной системе счисления, справа двоичное представление кода символа в UTF-8, одинаковыми цветами выделены разряды из кода Юникод и место в формате UTF-8 куда они были записаны).

Так как кодировка UTF-8 имеет переменную длину кода символа, необходимы служебные последовательности в байтах. Чтение данных происходит побайтно и считывая первый байт кода символа, можно однозначно определить количество байт, используемых для кодирования символа. Последовательность 10 в младших байтах позволяет выявить потери при передаче данных. Например по первому байту кода символа было определено, что символ кодируется тремя байтами (включая первый байт) поэтому ожидается, что два следующих байта будут иметь в старших разрядах значение 10, а байт следующий за ними будет начинаться или с 0 или с последовательности единиц (две или более единицы) и нуля после них. Если это не так – это говорит о возникшей ошибке при передаче или обработке информации.

Пример декодирования из UTF-8:

Дан неизвестный символ в кодировке UTF-8 – $E1899F_{16}$

1. Код символа в UTF-8 переводится в двоичную систему счисления $E1899F_{16}$ → $111000011000100110011111_2$
2. Так как код символа в двоичном представлении начинается с 1110 – это говорит о том, что для кодирования символа используется три байта и код в UTF-8 соответствует шаблону 1110xxxx 10xxxxxx 10xxxxxx, где x – значимый разряд кода в таблице Юникод.
3. Значимые разряды извлекаются из последовательности и записываются по порядку: 11100001 10001001 10011111 → 0001001001011111 → U+125F

UTF-16 – один из способов кодирования чисел при помощи шестнадцатибитных слов. При помощи этой кодировки можно кодировать символы в диапазонах от U+0000 до U+D7FF и от U+E000 до U+FFFF. Исключённый диапазон используется для символов, которые кодируются двумя шестнадцатибитными словами.

Если код символа в таблице Юникод лежит в диапазонах от U+0000 до U+D7FF и от U+E000 до U+FFFF, они представляются без изменений. В другом случае код символа арифметически сдвигается до нуля, т.е. из кода числа вычитается 10000_{16}. Старшие 10 бит результата суммируются с $D800_{16}$, а младшие 10 бит результата суммируются с $DC00_{16}$. Результат первого суммирования формирует старшие 16 бит кода, результат второго суммирования формирует младшие 16 бит кода.

Пример кодирования символа:

Дан символ с кодом U+E0110. Т.к. код символа больше чем U+FFFF, он кодируется двумя шестнадцатибитными словами.

1. Вычитаем из кода символа 10000_{16}. $E0110_{16}$ - 10000_{16} = $D0110_{16}$ = 11010000000100010000_2
2. Младшие 10 бит результата суммируем с $DC00_{16}$, а старшие 10 бит с $D800_{16}$.
   1101000000100010000 (цветами показаны старшие 10 бит и младшие 10 бит)
   1101000000 + $D800_{16}$ = 1101000000 + 1101100000000000 = 1101101101000000_2 = $DB40_{16}$
   0100010000 + $DC00_{16}$ = 0100010000 + 1101110000000000 = 1101 1101 0001 0000_2 = $DD10_{16}$
3. Результатом конкатенации шестнадцатибитных слов будет код символа в UTF-16
   $DB40DD10_{16}$.

При декодировании из кодировки UTF-16, если шестнадцатибитное слово не входит в исключенный диапазон, значит код символа в Юникод записан без изменений и декодирование как таковое не требуется. В противном случае символ закодирован двумя шестнадцатибитными словами и из первого нужно вычесть $D800_{16}$ и взять младшие 10 бит результата, а из следующего шестнадцатибитного слова вычесть $DC00_{16}$ и взять младшие 10 бит результата, и склеить их с результатом предыдущего вычитания. Затем прибавить к получившемуся числу 10000_{16}. Результатом будет являться код числа в таблице Юникод.

Пример декодирования символа из кодировки UTF-16, закодированного двуми шестнадцатибитными словами:

Дано два шестнадцатибитных слова $DBC0_{16}$ и $DFFF_{16}$.

1. Вычитаем из первого слова $D800_{16}$. $DBC0_{16}$ - $D800_{16}$ = $3C0_{16}$ = 001111000000_{16} (цветом выделены младшие 10 бит).
2. Вычитаем из второго слова $DC00_{16}$. $DFFF_{16}$ - $DC00_{16}$ = $3FF_{16}$ = 0011111111111_2 (цветом выделены младшие 10 бит).

3. Склеиваем младшие 10 бит результатов (сначала идёт 10 бит первого результата, затем 10 бит второго результата) – 1111000000111111111_2 = $F03FF_{16}$
4. Прибавляем к результату 10000_{16}. $F03FF_{16} + 10000_{16}$ = U+1003FF

UTF-32 – ещё один способ кодирования символов из Юникод, использующий для кодирования символов всегда 32 бита. Код символа в кодировке UTF-32 совпадает с кодом символа из таблицы Юникод. Главный недостаток этой кодировки – неэффективное использование памяти. Увеличение размера строк в кодировке UTF-32 по сравнению со строками в кодировках UTF-8 и UTF-16 не оправданы.