Krzaczki na ekranie

Tomasz IDZIASZEK

ASCII. Podstawowym sposobem reprezentacji znaków we współczesnych komputerach jest przyjęty w 1967 r. ASCII (czyt. aski, ang. *American Standard Code for Information Interchange*). Definiuje on 128 znaków, wśród których znajdują się 33 niedrukowalne znaki sterujące, znak odstępu, 52 litery (wielkie i małe litery alfabetu angielskiego), 10 cyfr i 32 znaki interpunkcyjne. ASCII przyporządkowuje każdemu znakowi liczbę z zakresu 0–127 (lub równoważnie szesnastkowo 0–7F). Poniższa tabelka przedstawia to przyporządkowanie:

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F	70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Pomimo że ASCII przypisuje znakom liczby 7-bitowe, w komputerze wygodnie jest przechowywać je w formacie jeden znak na bajt, ósmy bit zostawiając wyzerowany. Tak więc np. litera a ma przypisany kod 1100001_2, czyli będzie reprezentowana jako bajt 01100001. Zwróćmy też uwagę na kolejność znaków w ASCII. Wiele programów sortuje napisy, porównując kody ASCII znaków, co niekoniecznie musi być najlepszym sposobem – choć znak odstępu znajduje się przed innymi znakami, a litery są ułożone w kolejności alfabetycznej, to wszystkie wielkie litery znajdują się przed małymi.

Tajemnicze znaki sterujące wypełniające pierwszy wiersz tabeli nie są bezpośrednio wyświetlane, ale służą do sterowania sposobem wyświetlania tekstu. Większość z nich ma już wyłącznie znaczenie historyczne, w użyciu pozostają m.in.: NUL, który w wielu językach programowania oznacza koniec napisu, HT – przesunięcie w prawo do ustalonej kolumny (tabulacja), LF i CR – koniec wiersza, czy też ESC oznaczający wejście w specjalny tryb interpretowania znaków.

Na szczególną uwagę zasługują znaki sterujące oznaczające koniec wiersza. W pierwszych komputerach, które posługiwały się dalekopisami jako swoimi urządzeniami wejściowymi, koniec wiersza sygnalizowany był za pomocą pary znaków CR i LF. Pierwszy z nich instruował drukarkę, aby przesunęła głowicę do początku wiersza (ang. *carriage return*), natomiast drugi, aby wysunęła papier na następny wiersz (ang. *line feed*). Ta konwencja jest nadal w użyciu w systemie operacyjnym Windows oraz w wielu tekstowych protokołach internetowych. Jednakże np. w systemach Linux i Mac OS X koniec wiersza oznaczany jest pojedynczym znakiem LF. To może stanowić problem dla użytkowników, którzy przenoszą swoje pliki tekstowe pomiędzy komputerami działającymi pod kontrolą różnych systemów operacyjnych. Plik tekstowy zapisany pod systemem Linux będzie wyglądał w systemie Windows jak niepodzielony na wiersze, zaś plik zapisany w Windows będzie miał w Linuksie dodatkowe znaki na końcach wierszy. Wiele programów radzi sobie z obiema konwencjami, jednak nie wszystkie.

Ogonki. Pewnym problemem z ASCII jest to, że definiuje on jedynie litery alfabetu angielskiego. Pomysł, w jaki sposób pomóc nieanglojęzycznym użytkownikom komputerów, był dość naturalny: postanowiono rozszerzyć kodowanie do pełnych 8 bitów i wykorzystać nowe znaki do zakodowania symboli z alfabetów narodowych. I tak 128 dodatkowych znaków w zupełności wystarczyło dla języków wywodzących się z łaciny, a także dla języków, w których alfabet składa się z podobnej liczby liter (np. rosyjskiego czy greckiego). Jednak było to zdecydowanie za mało, żeby wszystkie te języki pomieścić jednocześnie. W związku z tym używano różnych konwencji na

Wraz z końcem lata, między 15 a 21 września, odbędą się w Krakowie coroczne

Międzynarodowe Warsztaty dla Młodych Matematyków

organizowane przez **Koło Matematyków Studentów UJ**. Tematem nadchodzących XVI Warsztatów będzie teoria ergodyczna i układy dynamiczne. Wśród „gwiazd" tego działu matematyki w tym roku jest zaproszony m.in. prof. Tomasz Downarowicz z Politechniki Wrocławskiej, specjalista z zakresu entropii w układach dynamicznych i entropii topologicznej.

Koło Matematyków zaprasza do udziału wszystkich zainteresowanych tematyką Warsztatów. Inicjatywa Koła obejmuje nie tylko zgłębianie wiedzy akademickiej, ale również rozmaite interakcje towarzyskie, w tym zwiedzanie miasta oraz atrakcje takie jak wyjście na kręgle, do teatru czy zwiedzanie krakowskiego ogrodu zoologicznego.

Aktualne informacje będą publikowane na stronie Warsztatów:

`http://kmsuj.im.uj.edu.pl/warsztaty`.

wykorzystanie dodatkowych znaków (tzw. *stron kodowych*), z których część została oficjalnymi standardami. I tak np. standard ISO-8859-1 (znany również jako Latin 1) definiuje litery dla większości języków zachodnioeuropejskich, natomiast ISO-8859-2 (Latin 2) dla języków środkowo- i wschodnioeuropejskich (w tym polskiego); ISO-8859-5 koduje cyrylicę, natomiast ISO-8859-7 – alfabet grecki.

Zanim jednak w 1993 r. Polski Komitet Normalizacyjny ustanowił normę PN-93 T-42118, która definiuje sposób kodowania polskich znaków zgodny z ISO-8859-2, producenci oprogramowania, którzy chcieli w swoich produktach używać polskich znaków, na własną rękę przypisywali im kody. Tak powstało około 20 różnych „standardów" o wdzięcznych nazwach, np. Mazovia, Cyfromat, CP852 czy CP1250. Ostatni z tych standardów nadal obowiązuje w systemie Windows jako domyślne kodowanie dla języków środkowoeuropejskich. Z tego powodu przenoszenie między różnymi systemami plików tekstowych, w których „ogonki" są zapisane w rozszerzonym ASCII, wymaga zastosowania odpowiedniej konwersji. Poniższa tabelka przedstawia, gdzie znajdują się polskie litery w dwóch najpopularniejszych kodowaniach oraz jakie znaki znajdują się na odpowiadających im pozycjach w kodowaniu Latin 1.

	8C	8F	9C	9F	A1	A3	A5	A6	AC	AF	B1	B3	B6	B9	BC	BF	C6	CA	D1	D3	E6	EA	F1	F3
Latin 2	⊘	⊘	⊘	⊘	Ą	Ł	Ľ	Ś	Ź	Ż	ą	ł	ś	š	ź	ż	Ć	Ę	Ń	Ó	ć	ę	ń	ó
CP1250	Ś	Ź	ś	ź	ˇ	Ł	Ą	¦	¬	Ż	±	ł	¶	ą	Ľ	ż	Ć	Ę	Ń	Ó	ć	ę	ń	ó
Latin 1	⊘	⊘	⊘	⊘	¡	£	¥	¦	¬	¯	±	³	¶	¹	¼	¿	Æ	Ê	Ñ	Ó	æ	ê	ñ	ó

O ile jeszcze pojedynczy użytkownik mógł wybrać ulubione kodowanie swojego języka i się go trzymać, o tyle sprawa stała się trudniejsza wraz z upowszechnieniem (i umiędzynarodowieniem) Internetu. Sprawę rozwiązano następująco: każda strona WWW powinna być zaopatrzona w informację, w jakim kodowaniu została przygotowana. Dzięki temu przeglądarka internetowa będzie mogła poprawnie zinterpretować tekst na tej stronie. Problem pojawiał się, gdy twórca strony nie opatrzył jej taką informacją, a rzeczywiste kodowanie strony było inne niż to, którego spodziewała się przeglądarka. W takiej sytuacji zamiast polskich znaków na ekranie widzieliśmy różne „krzaczki" (patrz rys. 1). Podobna sytuacja jest z mejlami – każda wiadomość musi zawierać informację o kodowaniu, w którym została wysłana.

	Latin 2	CP1250
Latin 2	ł ą c z n o ś ć	ł š c z n o ⊘ ć
CP1250	ł ± c z n o ¶ ć	ł ą c z n o ś ć
Latin 1	³ ± c z n o ¶ æ	³ ¹ c z n o ⊘ æ

Rys. 1. Tabelka przedstawia, co możemy zobaczyć na stronie internetowej poświęconej komunikacji, w zależności od tego, jak została ona zakodowana (kolumna) i jakiego kodowania oczekuje nasza przeglądarka (wiersz). Mogą pojawić się różnice w wyświetlaniu niezdefiniowanych znaków, w szczególności ostatnia pozycja w tabelce może także zostać wyświetlona jako ³ ¹ c z n o œ æ, jeśli zamiast kodowania Latin 1 przeglądarka użyje jego nadzbioru – kodowania CP1252 (stosowanego w systemie Windows).

Unikod. Mnogość stron kodowych nadal nie rozwiązywała dwóch poważnych problemów: nie było możliwości zakodowania w tym samym dokumencie tekstu w językach o rozłącznych alfabetach (np. polskim i rosyjskim), jak również tekstów w językach o dużej liczbie znaków (50 tysięcy chińskich znaków nijak nie dało się zmieścić w 8 bitach). Naturalnym pomysłem na rozwiązanie tych problemów jest zrezygnowanie z ograniczenia „jeden bajt na znak" i zapisywanie każdego znaku np. w dwóch bajtach. Bezpośrednia implementacja tego rozwiązania ma jednak szereg wad, wśród których najbardziej oczywistą jest to, że każdy tekst zwiększyłby dwukrotnie zapotrzebowanie na pamięć komputera.

W 1987 r. rozpoczęto prace nad uniwersalnym międzynarodowym sposobem reprezentacji znaków, który został nazwany Unikodem (ang. *Unicode*). Jest on w ciągłym rozwoju, we wrześniu 2012 r. ogłoszona została wersja 6.2 tego standardu. Unikod definiuje 1 114 112 znaków (w przedziale 0–10FFFF). Z uwagi na ogromną ich liczbę w Unikodzie swoje miejsce znalazły nie tylko wszystkie używane współcześnie alfabety, lecz także historyczne (jak egipskie hieroglify czy pismo klinowe). Ponadto występuje w nim wiele symboli – od matematycznych, przez muzyczne, aż po całkowicie niszowe (rys. 2).

Znak	Liczba	Znak	Liczba
ą	U+0105	€	U+20AC
Ω	U+03A9	⫋	U+228A
Я	U+042F	♂	U+2642
א	U+05D0	♣	U+2663
有	U+6709	♻	U+267B
ə	U+0259	♪	U+1D160
𑱼	U+1245C	🐰	U+270C

Rys. 2. Przykładowe znaki zdefiniowane w Unikodzie.

Jedną z kluczowych decyzji projektowych różniących ASCII od Unikodu jest wprowadzenie rozróżnienia między tym, w jaki sposób danemu znakowi przypisujemy unikalną liczbę, a tym, w jaki sposób tę liczbę będziemy zapisywać (kodować) w postaci ciągu bitów. Liczbę przypisaną znakowi w Unikodzie będziemy pisać szesnastkowo z prefiksem U+. Pierwsze 128 liczb jest przypisanych tak samo jak w ASCII, zatem np. A ma przypisaną liczbę U+0041.

Drugą kwestię regulują konkretne kodowania, spośród których najpopularniejsze jest UTF-8 (ang. *8-bit Unicode Transformation Format*). Głównym pomysłem jest zrezygnowanie z ustalonej liczby bajtów na znak. Jeśli przypisana znakowi liczba jest mniejsza niż 128, to kodujemy go w jednym bajcie. W przeciwnym przypadku znak będzie zakodowany w dwóch do czterech bajtach, według przepisu z tabelki.

Zakres oraz liczba bitów		Bajt 1	Bajt 2	Bajt 3	Bajt 4
0 – 7F	7	0*xxxxxx*			
80 – 7FF	11	110*xxxx*	10*xxxxxx*		
800 – FFFF	16	1110*xxx*	10*xxxxxx*	10*xxxxxx*	
10000 – 1FFFF	21	11110*xx*	10*xxxxxx*	10*xxxxxx*	10*xxxxxx*

Przykładowo, litera ą ma kod U+105, zatem w systemie binarnym będzie to 100000101_2. Do zapisu tej liczby potrzebujemy 9 bitów, więc zakodujemy ją w dwóch bajtach następująco: 11000100 10000101. Z kolei chiński znak 大 ma kod U+5927, czyli 101100100100111_2 binarnie, zatem potrzebujemy dla niego 15 bitów; w trzech bajtach będzie to 11100101 10100100 10100111.

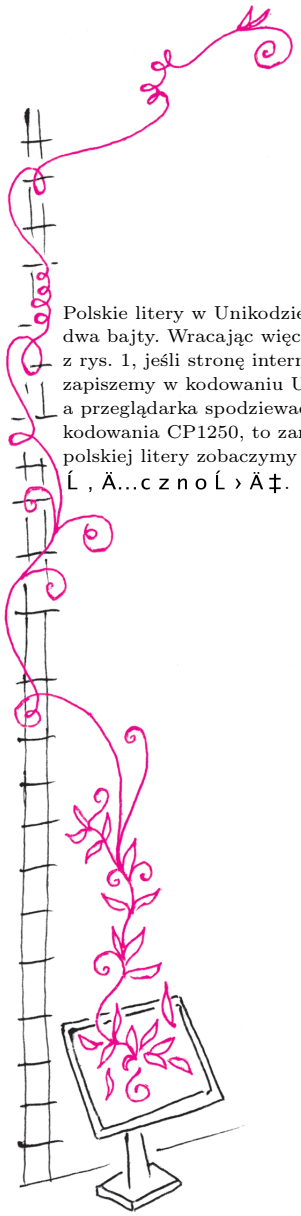
Kodowanie UTF-8 ma sporo zalet (których nie ma np. kodowanie za pomocą dwóch bajtów). Przede wszystkim jest wstecznie kompatybilne z ASCII – każdy napis, w którym ósmy bit wszystkich bajtów jest wyzerowany, jest poprawnym (i tym samym) napisem w obu kodowaniach. Dzięki temu teksty w języku angielskim w UTF-8 nie zajmują więcej miejsca niż w ASCII. Ponadto, jeśli w napisie znajdzie się bajt 0, to nie może on być częścią jakiegoś wielobajtowego znaku, tylko zawsze koduje znak NUL – dzięki czemu zachowana jest kompatybilność z językami programowania, które tym znakiem oznaczają koniec napisu. Zadbano również o to, abyśmy – po rozpoczęciu czytania tekstu w losowym miejscu – byli w stanie łatwo stwierdzić, gdzie zaczyna się kod kolejnego znaku (pierwszy bajt, który nie jest postaci 10*xxxxxx*).

Wadą kodowania UTF-8 jest to, że analiza tekstu zakodowanego UTF-8 jest trudniejsza niż w przypadku ASCII. Dla przykładu, stwierdzenie, ile liter ma napis, wymaga przeczytania całego ciągu bajtów. Niemniej jednak jest to niewielka cena jak na to, że Unikod zdejmuje z nas ból głowy spowodowany koniecznością obsługi wielu stron kodowych.

Zagrożenia. Choć mówią, że od przybytku głowa nie boli, warto jednak wspomnieć o pewnych kłopotach, które może powodować mnogość znaków Unikodu. Zagadka dla Czytelników: proszę wskazać różnice między trzema znakami: A, A i A. Są to kolejno litery alfabetu łacińskiego (U+0041), greckiego (U+0391) i cyrylicy (U+0410). Choć wyglądają tak samo, to Unikod przypisuje im różne liczby, co można wykorzystać. Przykładowo, pierwsze systemy, które miały za zadanie wykrywać plagiaty, można było łatwo oszukać, zamieniając w pracy niektóre wystąpienia liter łacińskich na ich wizualnie identyczne odpowiedniki w cyrylicy.

Należy też uważać na łącza internetowe, które klikamy. Od 2010 r. jest możliwość rejestracji domen o nazwach zawierających dowolne znaki Unikodu (zobacz IDN, ang. *Internationalized Domain Name*). Może się okazać, że nie wchodzimy na stronę naszego banku, ale na odpowiednio spreparowaną stronę o nazwie łudząco podobnej do nazwy domeny banku.

Unikod jest dość skomplikowany i w tym krótkim artykule nie byliśmy w stanie opisać wszystkich jego zalet i wad. Żeby uczynić go prawdziwie międzynarodowym, jego projektanci musieli zmierzyć się z wieloma problemami, bardziej złożonymi niż brak polskich „ogonków”. W szczególności niektóre języki zapisujemy od strony prawej do lewej. Dzięki Unikodowi możemy w jednym dokumencie połączyć tekst angielski z arabskim. Na koniec trzeba wspomnieć, że aby w pełni korzystać z dobrodziejstw Unikodu, należy zainstalować na komputerze krój pisma, który będzie zawierał potrzebne nam znaki. Zaprojektowanie kroju z pełnym wsparciem dla Unikodu to tytaniczna praca, więc projektanci z reguły ograniczają się do wybranego podzbioru znaków.



Polskie litery w Unikodzie zajmują po dwa bajty. Wracając więc do przykładu z rys. 1, jeśli stronę internetową zapiszemy w kodowaniu UTF-8, a przeglądarka spodziewać się będzie kodowania CP1250, to zamiast każdej polskiej litery zobaczymy dwa „krzaczki”: Ĺ , Ä…c z n o Ĺ › Ä‡.

Fakt, że Unikod radzi sobie z oboma kierunkami pisma, może rodzić pole do kolejnych subtelnych nadużyć. Unikod definiuje znak sterujący RLO (U+202E), który wymusza przełączenie w tryb wyświetlania tekstu od strony prawej do lewej. System Windows obsługuje nazwy plików w Unikodzie. Jeśli więc stworzymy plik o nazwie a n n RLO t x t . e x e, to użytkownikowi zostanie zaprezentowana niewinnie wyglądająca nazwa a n n e x e . t x t. Kliknięcie na nazwę pliku nie otworzy edytora tekstu, ale uruchomi (potencjalnie złośliwy) program.