Wstęp do informatyki

Reprezentacja danych

Piotr Fulmański

Wydział Matematyki i Informatyki,
Uniwersytet Łódzki, Polska

January 9, 2020

Spis treści

Niezbędne wiadomości ogólne

Bit

Bit (w ang. kawałek, skrót od *binary digit*, czyli *cyfra dwójkowa*) to najmniejsza ilość informacji potrzebna do określenia, który z dwóch równie prawdopodobnych stanów przyjął układ. Bit jest jednostką logiczną.

Bajt

Bajt[a] – najmniejsza adresowalna jednostka informacji pamięci komputerowej, składająca się z bitów.

[a] Uwaga lingwistyczna: w dopełniaczu poprawną formą jest zarówno *bajtu* jak i *bajta*.

W praktyce przyjmuje się, że jeden bajt to 8 bitów. Ponieważ nie wynika to z podanej definicji, więc jednostkę składającą się z ośmiu bitów nazywa się również **oktetem**[1]. Bywa też że bajt definiuje się jako najmniejszą adresowalną jednostkę pamięci (oznaczaną *char*, gdyż wystarczała do zakodowania pojedyńczego znaku) złożoną z 8 bitów. W starszych maszynach nie stosowano pojęcia bajt ani oktet, najmniejszą jednostką było **słowo maszynowe** składające się z pewnej, zależnej od architektury, ilości bitów.

[1] Warto zauważyć, iż np. w języku francuskim nie używa się pojęcia *bajt*, ale właśnie *oktet*.

Bajt

Bajt[a] – najmniejsza adresowalna jednostka informacji pamięci komputerowej, składająca się z bitów.

[a] Uwaga lingwistyczna: w dopełniaczu poprawną formą jest zarówno *bajtu* jak i *bajta*.

W praktyce przyjmuje się, że jeden bajt to 8 bitów. Ponieważ nie wynika to z podanej definicji, więc jednostkę składającą się z ośmiu bitów nazywa się również **oktetem**[1]. Bywa też że bajt definiuje się jako najmniejszą adresowalną jednostkę pamięci (oznaczaną *char*, gdyż wystarczała do zakodowania pojedyńczego znaku) złożoną z 8 bitów. W starszych maszynach nie stosowano pojęcia bajt ani oktet, najmniejszą jednostką było **słowo maszynowe** składające się z pewnej, zależnej od architektury, ilości bitów.

[1] Warto zauważyć, iż np. w języku francuskim nie używa się pojęcia *bajt*, ale właśnie *oktet*.

Niezbędne wiadomości ogólne

Jednostki – zwyczajowe nazwy

- **byte** = 8 bits
- **nibble (nybble, nyble, half an octet, half-byte, tetrade)** = 4 bits
- **char** = 8 bits
- **word** = 16 bits
- **dword (double word)** = 32 bits
- **qword (quad word)** = 64 bits

MSB

Najbardziej znaczący bit (ang. *most significant bit*, MSB), zwany też **najstarszym bitem** to bit o największej wadze (przedstawiający największą wartość liczbową), zwykle znajdujący się w słowie cyfrowym na miejscu najbardziej wysuniętym w lewo.

LSB

Najmniej znaczący bit (ang. *least significant bit*, LSB), zwany też **najmłodszym bitem** to bit o najmniejszej wadze (przedstawiający najmniejszą wartość liczbową), zwykle znajdujący się w słowie cyfrowym na miejscu najbardziej wysuniętym w prawo. W naturalnym kodzie binarnym świadczy o parzystości liczby.

Przedrostki dziesiętne i binarne

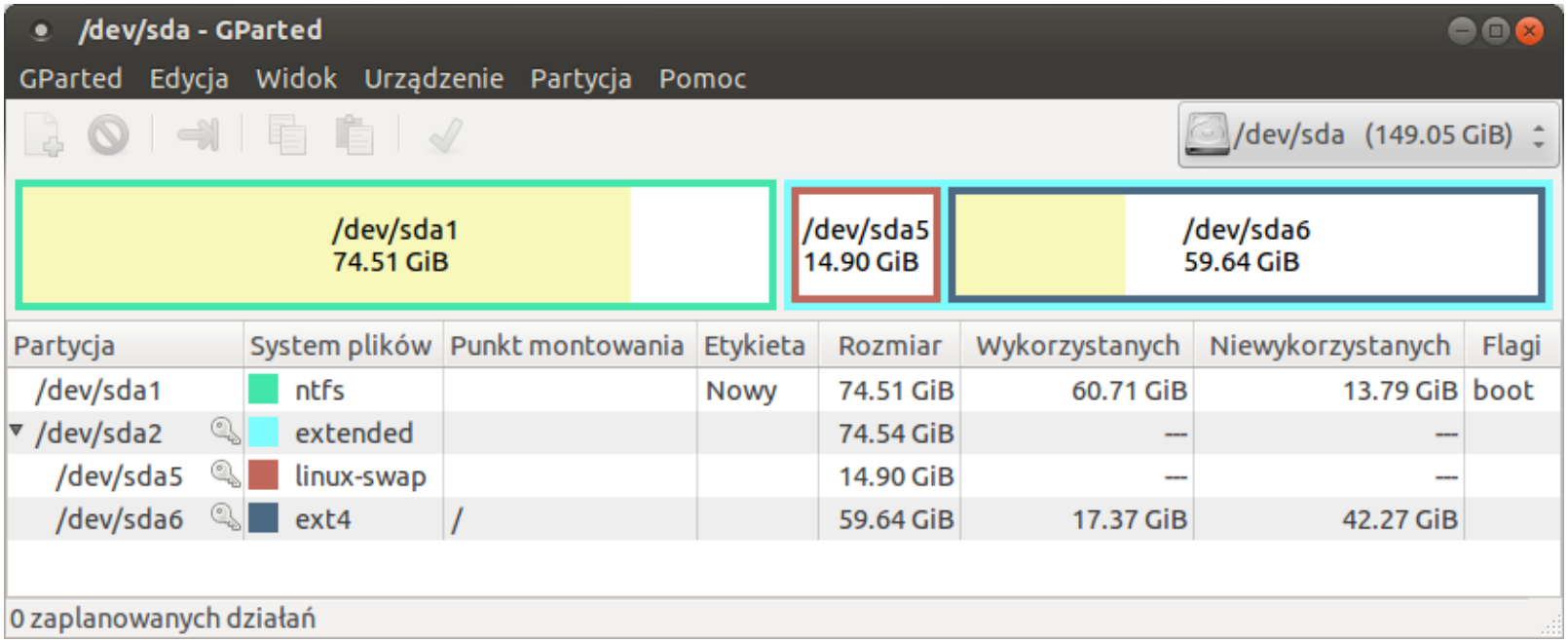
Stosowanie przedrostków *kilo*, *mega*, *giga* itd. do określania odpowiednich potęg liczby dwa jest niezgodne z wytycznymi układu SI (np. kilo oznacza 1000, a nie 1024). W celu odróżnienia przedrostków o mnożniku 1000 od przedrostków o mnożniku 1024, w styczniu 1997 pojawiła się propozycja ujednoznacznienia opracowana przez Międzynarodową Komisję Elektrotechniczną (ang. *International Electrotechnical Commission*, skrót IEC) polegająca na **dodawaniu litery „*i*" po symbolu przedrostka dwójkowego, oraz „*bi*" po jego nazwie**. Nowe przedrostki nazywane zostały **przedrostkami dwójkowymi** (binarnymi).

Przedrostki dziesiętne i binarne

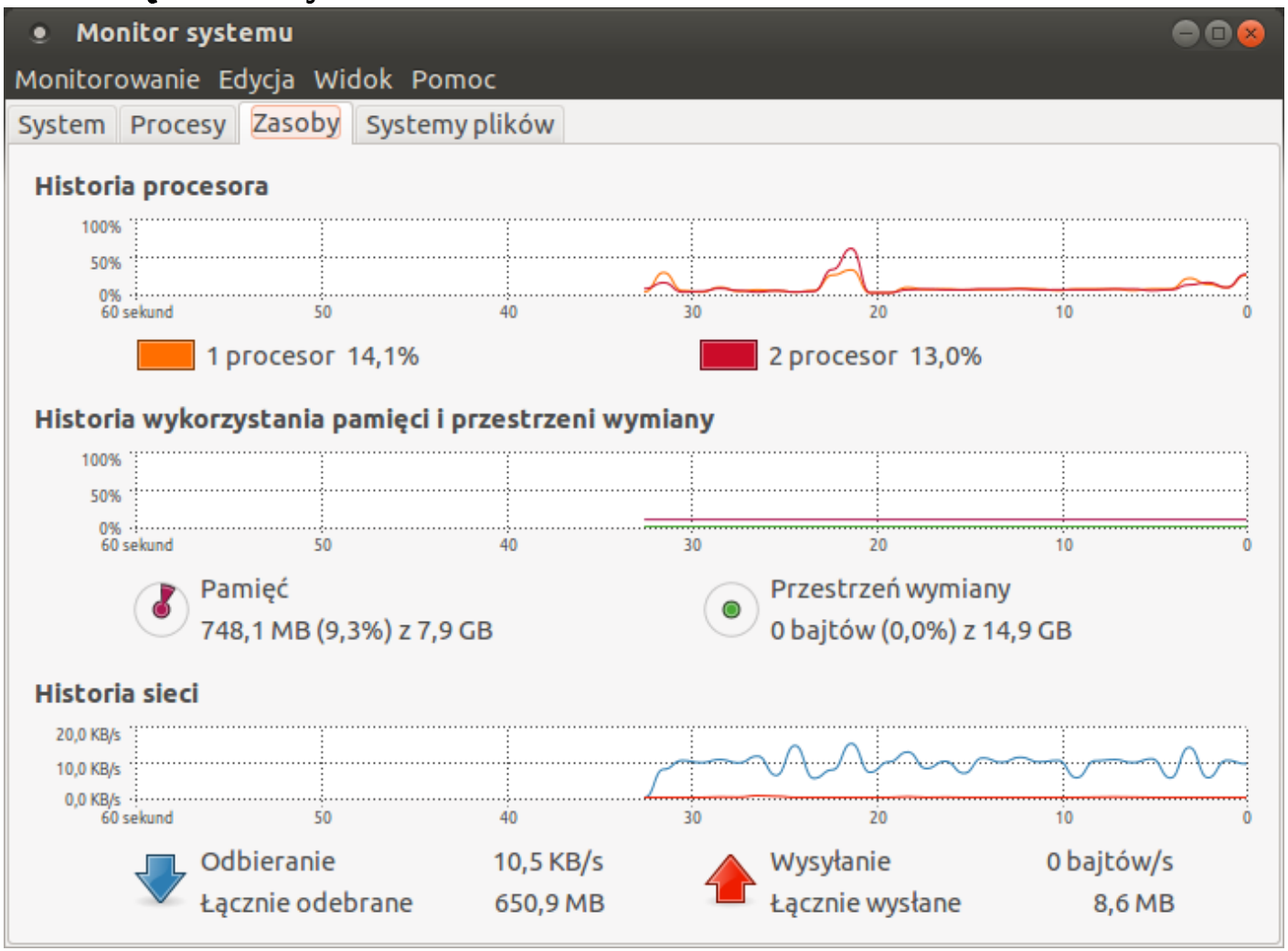
Wielokrotności bitów					
Przedrostki dziesiętne (SI)			Przedrostki binarne (IEC 60027-2)		
Nazwa	Symbol	Mnożnik	Nazwa	Symbol	Mnożnik
kilo	kb	$10^3 = 1000^1$	kibi	Kib	$2^{10} = 1024^1$
mega	Mb	$10^6 = 1000^2$	mebi	Mib	$2^{20} = 1024^2$
giga	Gb	$10^9 = 1000^3$	gibi	Gib	$2^{30} = 1024^3$
tera	Tb	$10^{12} = 1000^4$	tebi	Tib	$2^{40} = 1024^4$
peta	Pb	$10^{15} = 1000^5$	pebi	Pib	$2^{50} = 1024^5$
eksa	Eb	$10^{18} = 1000^6$	eksbi	Eib	$2^{60} = 1024^6$
zetta	Zb	$10^{21} = 1000^7$	zebi	Zib	$2^{70} = 1024^7$
jotta	Yb	$10^{24} = 1000^8$	jobi	Yib	$2^{80} = 1024^8$

Tak więc mamy:
1KiB (jeden kibibajt) = 1024 bajty
1Kib (jeden kibibit) = 1024 bity = 128 bajty

Większość nowoczesnych systemów operacyjnych i aplikacji stosuje już nowe nazewnictwo



W związku z tym czasem nie wiadomo o co chodzi



Niezbędne wiadomości ogólne

Przedrostki dziesiętne i binarne

```
eth0      Link encap:Ethernet  HWaddr 00:24:8c:36:3c:24
          inet addr:10.0.80.18  Bcast:10.0.255.255  Mask:255.255.0.0
          inet6 addr: fe80::224:8cff:fe36:3c24/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:8607197 errors:0 dropped:0 overruns:0 frame:0
          TX packets:91626 errors:0 dropped:0 overruns:0 carrier:11
          collisions:0 txqueuelen:1000
          RX bytes:685816320 (685.8 MB)  TX bytes:9026642 (9.0 MB)
          Interrupt:17
```

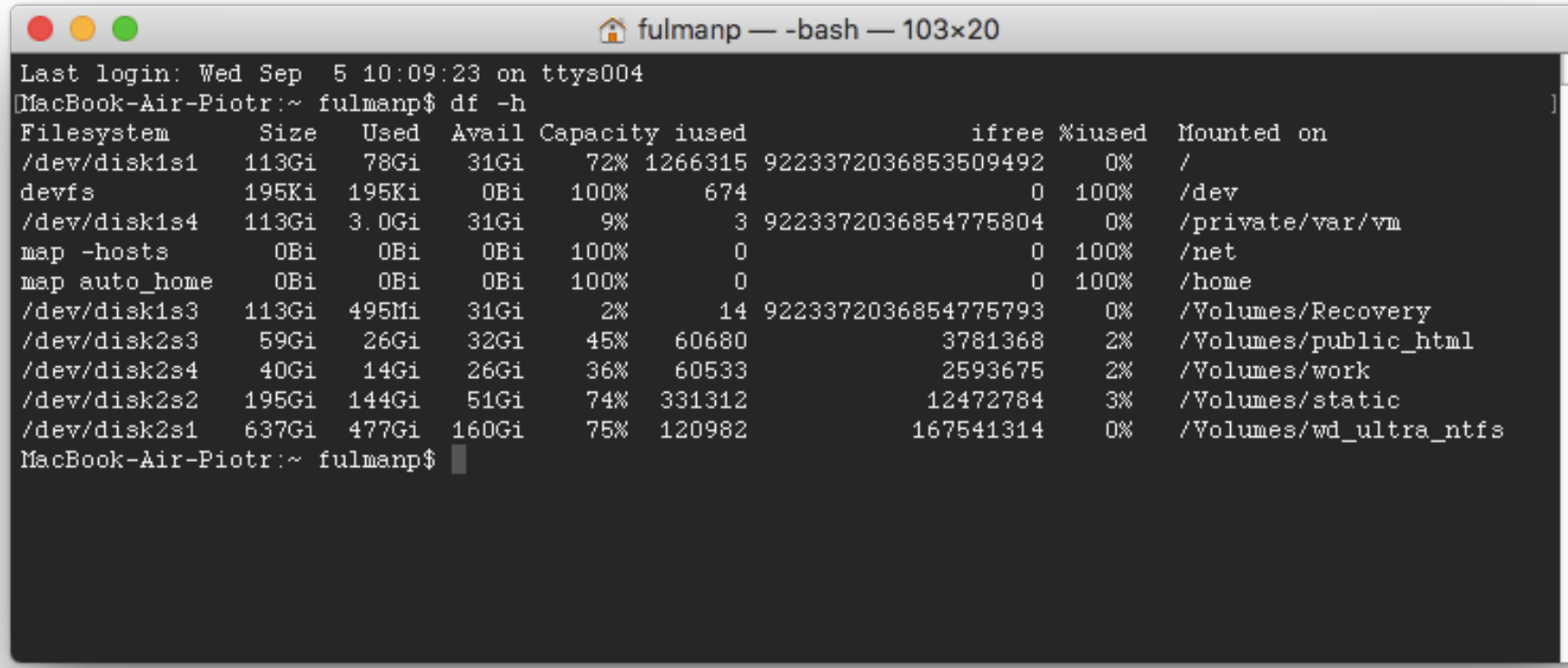
```
head is a program on Unix like systems used to display the beginning
of a text file or a stream of data. One of the options is

   -c [-]K print the first K bytes of each file; with the leading -,
           print all but the last K bytes of each file.

K may have a multiplier suffix
    b 512,
    kB 1000, K 1024,
    MB 1000*1000, M 1024*1024,
    GB 1000*1000*1000, G 1024*1024*1024,
    and so on for T, P, E, Z, Y.
```

Niezbędne wiadomości ogólne

Przedrostki dziesiętne i binarne



Endianess – kolejność bajtów

W sytuacjach, gdy dane zapisywane są przy użyciu wielu (przynajmniej dwóch) bajtów, nie istnieje jeden unikalny sposób uporządkowania tych bajtów w pamięci lub w czasie transmisji przez dowolne medium. W związku z tym dodatkowo określić trzeba zgodnie z jaką konwencją ustalana jest kolejność bajtów (ang. *byte order* lub *endianness*). Jest to sytuacja analogiczna do zapisu pozycyjnego liczb lub kierunku pisma w różnych językach.

Endianess – kolejność bajtów

Big endian

Big endian (BE, z j. ang. dosłownie oznacza *grubokońcowość*) to forma zapisu danych, w której najbardziej znaczący bajt (nazywany także wysokim bajtem, z ang. *high-order byte*) umieszczony jest jako pierwszy. Jest ona analogiczna do używanego na co dzień sposobu zapisu liczb. Procesor zapisujący 32-bitowe wartości w pamięci, przykładowo 1A2B3C4D pod adresem 100, umieszcza dane, zajmując adresy od 100 do 103 w następującej kolejności: 1A, 2B, 3C, 4D.

Endianess – kolejność bajtów

Little endian

Little endian (LE, z j. ang. dosłownie oznacza *cienkokońcowość*) to forma zapisu danych, w której najmniej znaczący bajt (zwany też dolnym bajtem, z ang. *low-order byte*) umieszczony jest jako pierwszy. Jest ona odwrotna do używanego na co dzień sposobu zapisu liczb. Procesor zapisujący 32-bitowe wartości w pamięci, przykładowo 1A2B3C4D pod adresem 100, umieszcza dane zajmując adresy od 100 do 103 w następującej kolejności: 4D, 3C, 2B, 1A.

Zapamiętać!

- Wszelka informacja przetwarzana przez system komputerowy jest ciągiem zer i jedynek. Niczym więcej.
- Z punktu widzenia systemu KAŻDA informacja to strumień zer i jedynek.
- Ten sam ciąg zer i jedynek raz może być zdjęciem naszego przyjaciela innym razem naszą ulubioną MP3 a jeszcze innym razem listem do cioci.
- To my, czyli użytkownik, mówimy jak interpretować dany ciąg zer i jedynek.
- Od sposobu interpretacji zależy co tak naprawdę odczytamy.
- To nie plik graficzny informuje nas o tym, że jest plikiem graficznym, ale to my plik interpretujemy jak gdyby był plikiem graficznym.[a]

[a] Oczywiście większość współczesnych plików zawiera w sobie informacje o przenoszonych danych, ale jest to tylko i wyłącznie po to aby ułatwić życie.

Zapamiętać!

- **Wszelka informacja przetwarzana przez system komputerowy jest ciągiem zer i jedynek. Niczym więcej.**
- Z punktu widzenia systemu KAŻDA informacja to strumień zer i jedynek.
- Ten sam ciąg zer i jedynek raz może być zdjęciem naszego przyjaciela innym razem naszą ulubioną MP3 a jeszcze innym razem listem do cioci.
- To my, czyli użytkownik, mówimy jak interpretować dany ciąg zer i jedynek.
- Od sposobu interpretacji zależy co tak naprawdę odczytamy.
- To nie plik graficzny informuje nas o tym, że jest plikiem graficznym, ale to my plik interpretujemy jak gdyby był plikiem graficznym.[a]

[a] Oczywiście większość współczesnych plików zawiera w sobie informacje o przenoszonych danych, ale jest to tylko i wyłącznie po to aby ułatwić życie.

Zapamiętać!

- **Wszelka informacja przetwarzana przez system komputerowy jest ciągiem zer i jedynek. Niczym więcej.**
- **Z punktu widzenia systemu KAŻDA informacja to strumień zer i jedynek.**
- Ten sam ciąg zer i jedynek raz może być zdjęciem naszego przyjaciela innym razem naszą ulubioną MP3 a jeszcze innym razem listem do cioci.
- To my, czyli użytkownik, mówimy jak interpretować dany ciąg zer i jedynek.
- Od sposobu interpretacji zależy co tak naprawdę odczytamy.
- To nie plik graficzny informuje nas o tym, że jest plikiem graficznym, ale to my plik interpretujemy jak gdyby był plikiem graficznym.[a]

[a] Oczywiście większość współczesnych plików zawiera w sobie informacje o przenoszonych danych, ale jest to tylko i wyłącznie po to aby ułatwić życie.

Zapamiętać!

- **Wszelka informacja przetwarzana przez system komputerowy jest ciągiem zer i jedynek. Niczym więcej.**
- **Z punktu widzenia systemu KAŻDA informacja to strumień zer i jedynek.**
- **Ten sam ciąg zer i jedynek raz może być zdjęciem naszego przyjaciela innym razem naszą ulubioną MP3 a jeszcze innym razem listem do cioci.**
- To my, czyli użytkownik, mówimy jak interpretować dany ciąg zer i jedynek.
- Od sposobu interpretacji zależy co tak naprawdę odczytamy.
- To nie plik graficzny informuje nas o tym, że jest plikiem graficznym, ale to my plik interpretujemy jak gdyby był plikiem graficznym.[a]

[a] Oczywiście większość współczesnych plików zawiera w sobie informacje o przenoszonych danych, ale jest to tylko i wyłącznie po to aby ułatwić życie.

Zapamiętać!

- **Wszelka informacja przetwarzana przez system komputerowy jest ciągiem zer i jedynek. Niczym więcej.**
- **Z punktu widzenia systemu KAŻDA informacja to strumień zer i jedynek.**
- **Ten sam ciąg zer i jedynek raz może być zdjęciem naszego przyjaciela innym razem naszą ulubioną MP3 a jeszcze innym razem listem do cioci.**
- **To my, czyli użytkownik, mówimy jak interpretować dany ciąg zer i jedynek.**
- **Od sposobu interpretacji zależy co tak naprawdę odczytamy.**
- **To nie plik graficzny informuje nas o tym, że jest plikiem graficznym, ale to my plik interpretujemy jak gdyby był plikiem graficznym.**[a]

[a] Oczywiście większość współczesnych plików zawiera w sobie informacje o przenoszonych danych, ale jest to tylko i wyłącznie po to aby ułatwić życie.

Zapamiętać!

- **Wszelka informacja przetwarzana przez system komputerowy jest ciągiem zer i jedynek. Niczym więcej.**
- **Z punktu widzenia systemu KAŻDA informacja to strumień zer i jedynek.**
- **Ten sam ciąg zer i jedynek raz może być zdjęciem naszego przyjaciela innym razem naszą ulubioną MP3 a jeszcze innym razem listem do cioci.**
- **To my, czyli użytkownik, mówimy jak interpretować dany ciąg zer i jedynek.**
- **Od sposobu interpretacji zależy co tak naprawdę odczytamy.**
- **To nie plik graficzny informuje nas o tym, że jest plikiem graficznym, ale to my plik interpretujemy jak gdyby był plikiem graficznym.**[a]

[a] Oczywiście większość współczesnych plików zawiera w sobie informacje o przenoszonych danych, ale jest to tylko i wyłącznie po to aby ułatwić życie.

Zapamiętać!

- **Wszelka informacja przetwarzana przez system komputerowy jest ciągiem zer i jedynek. Niczym więcej.**
- **Z punktu widzenia systemu KAŻDA informacja to strumień zer i jedynek.**
- **Ten sam ciąg zer i jedynek raz może być zdjęciem naszego przyjaciela innym razem naszą ulubioną MP3 a jeszcze innym razem listem do cioci.**
- **To my, czyli użytkownik, mówimy jak interpretować dany ciąg zer i jedynek.**
- **Od sposobu interpretacji zależy co tak naprawdę odczytamy.**
- **To nie plik graficzny informuje nas o tym, że jest plikiem graficznym, ale to my plik interpretujemy jak gdyby był plikiem graficznym.**[a]

[a] Oczywiście większość współczesnych plików zawiera w sobie informacje o przenoszonych danych, ale jest to tylko i wyłącznie po to aby ułatwić życie.

Kilka analogii

Analogia językowa

Co oznacza słowo: ***para***

- jeśli wiemy, że jest to słowo języka polskiego, to: ***dwa obiekty***;
- jeśli wiemy, że jest to słowo języka hiszpańskiego, to: ***dla***.

Analogia liczbowa

Liczba ***osiem*** może być zapisana jako

- ***8*** w dziesiętnym systemie liczbowym;
- ***VIII*** w rzymskim systemie liczbowym;
- ***1000*** w dwójkowym systemie liczbowym.

Kilka analogii

Analogia językowa

Co oznacza słowo: ***para***

- jeśli wiemy, że jest to słowo języka polskiego, to: ***dwa obiekty***;
- jeśli wiemy, że jest to słowo języka hiszpańskiego, to: ***dla***.

Analogia liczbowa

Liczba ***osiem*** może być zapisana jako

- ***8*** w dziesiętnym systemie liczbowym;
- ***VIII*** w rzymskim systemie liczbowym;
- ***1000*** w dwójkowym systemie liczbowym.

Kilka analogii

Analogia językowa

Co oznacza słowo: ***para***

- jeśli wiemy, że jest to słowo języka polskiego, to: ***dwa obiekty***;
- jeśli wiemy, że jest to słowo języka hiszpańskiego, to: ***dla***.

Analogia liczbowa

Liczba ***osiem*** może być zapisana jako

- ***8*** w dziesiętnym systemie liczbowym;
- ***VIII*** w rzymskim systemie liczbowym;
- ***1000*** w dwójkowym systemie liczbowym.

Kilka analogii

Analogia językowa

Co oznacza słowo: ***para***

- jeśli wiemy, że jest to słowo języka polskiego, to: ***dwa obiekty***;
- jeśli wiemy, że jest to słowo języka hiszpańskiego, to: ***dla***.

Analogia liczbowa

Liczba ***osiem*** może być zapisana jako

- ***8*** w dziesiętnym systemie liczbowym;
- ***VIII*** w rzymskim systemie liczbowym;
- ***1000*** w dwójkowym systemie liczbowym.

Kilka analogii

Analogia językowa

Co oznacza słowo: ***para***

- jeśli wiemy, że jest to słowo języka polskiego, to: ***dwa obiekty***;
- jeśli wiemy, że jest to słowo języka hiszpańskiego, to: ***dla***.

Analogia liczbowa

Liczba ***osiem*** może być zapisana jako

- ***8*** w dziesiętnym systemie liczbowym;
- ***VIII*** w rzymskim systemie liczbowym;
- ***1000*** w dwójkowym systemie liczbowym.

Kilka analogii

Analogia językowa

Co oznacza słowo: ***para***

- jeśli wiemy, że jest to słowo języka polskiego, to: ***dwa obiekty***;
- jeśli wiemy, że jest to słowo języka hiszpańskiego, to: ***dla***.

Analogia liczbowa

Liczba ***osiem*** może być zapisana jako

- ***8*** w dziesiętnym systemie liczbowym;
- ***VIII*** w rzymskim systemie liczbowym;
- ***1000*** w dwójkowym systemie liczbowym.

Co kodujemy?

O kodowaniu czego mówić będziemy

- liczby naturalne
- liczby całkowite
- liczby rzeczywiste
- znaki
- plik graficzny `bmp`
- pakiet TCP/IP

Kodowanie liczb naturalnych

Naturalny zapis wykorzystywany do zapisu liczby w dwójkowym systemie liczbowym.

Kodowanie liczb całkowitych

- znak-moduł
- uzupełnieniowa do dwóch (U2)

Kodowanie liczb całkowitych

- znak-moduł
- uzupełnieniowa do dwóch (U2)

```
00101   =   +5 (zm)
10001   =   -1 (zm)
====== +
10110   =   -6 (zm) ??!!
```

Uzupełnienie dwójkowe

Uzupełnieniem dwójkowym liczby x, zapisanej za pomocą n bitów, nazywamy liczbę $x_{U2} = 2^n - x$.

Idea

$$x = 1011_2$$
$$(x)_{U2} = 10000_2 - 1011_2 = 0101_2$$
$$y = (x)_{U2}$$
$$(y)_{U2} = 10000_2 - 0101_2 = 1011_2$$
$$x = 1011_2 = (y)_{U2} = ((x)_{U2})_{U2}$$

Uzupełnienie dwójkowe

Uzupełnieniem dwójkowym liczby x, zapisanej za pomocą n bitów, nazywamy liczbę $x_{U2} = 2^n - x$.

Idea

$$
\begin{aligned}
x &= 1011_2 \\
(x)_{U2} &= 10000_2 - 1011_2 = 0101_2 \\
y &= (x)_{U2} \\
(y)_{U2} &= 10000_2 - 0101_2 = 1011_2 \\
x = 1011_2 = (y)_{U2} &= ((x)_{U2})_{U2}
\end{aligned}
$$

Liczby rzeczywiste

Kodowanie liczb rzeczywistych

- zapis stałoprzecinkowy
- zapis zmiennoprzecinkowy

Liczby rzeczywiste

Zapis stałoprzecinkowy

Zapis zmiennoprzecinkowy

$$z_m M \cdot 2^{z_c C}$$

Zapis zmiennoprzecinkowy: normalizacja

$$1.0 \leq M < 2.0$$

Zapis zmiennoprzecinkowy – przykład

Przyjmujemy następujące założenia

- wykorzystujemy 8 bitów;
- pierwszy bit od lewej (7) oznacza znak liczby;
- bity (6-4) oznaczają mantysę;
- bity (3-0) oznaczają cechę;
- stała K_C przyjmuje wartość 7.

Zapis zmiennoprzecinkowy

$$z_m M \cdot 2^{z_c C}$$

Zapis zmiennoprzecinkowy: normalizacja

$$1.0 \leq M < 2.0$$

Zapis zmiennoprzecinkowy – przykład

Przyjmujemy następujące założenia

- wykorzystujemy 8 bitów;
- pierwszy bit od lewej (7) oznacza znak liczby;
- bity (6-4) oznaczają mantysę;
- bity (3-0) oznaczają cechę;
- stała K_C przyjmuje wartość 7.

Zapis zmiennoprzecinkowy

$$z_m M \cdot 2^{z_c C}$$

Zapis zmiennoprzecinkowy: normalizacja

$$1.0 \leq M < 2.0$$

Zapis zmiennoprzecinkowy – przykład

Przyjmujemy następujące założenia

- wykorzystujemy 8 bitów;
- pierwszy bit od lewej (7) oznacza znak liczby;
- bity (6-4) oznaczają mantysę;
- bity (3-0) oznaczają cechę;
- stała K_C przyjmuje wartość 7.

- Liczbę pojedynczej precyzji w formacie IEEE-754 zapisujemy za pomocą **trzydziestu dwóch bitów**.
- Pierwszym bitem jest bit znaku S (sign). Jeśli liczba jest ujemna, S przyjmie wartość 1, jeśli dodatnia, S jest równe zero.
- Dalej następuje 8 bitów kodujących wykładnik (cechę) z wartością BIAS=127 co daje zakres wykładników $[-127, 128]$, przy czym najmniejsza i największa wartość wykładnika ma znaczenie specjalne.
- Kolejne 23 bity to mantysa liczby, przy czym pomija się wiodący, niezerowy bit. Daje to około 7–8 dziesiętnych miejsc znaczących i zakres od około $\pm 1.18 \cdot 10^{-38}$ do około $\pm 3.4 \cdot 10^{38}$.

Kodowanie

Kodowaniem nazwiemy proces zamiany znaku wpisanego z klawiatury lub innego urządzenia wczytującego na jego reprezentację cyfrową, czyli zapisanie jego przy pomocy ciągu zer i jedynek.

Kod PF1

Przyjmujemy następujący sposób kodowania znaków alfanumerycznych

a	b	c	d	e	f	g	h	i	j	k	l	m	n
0	1	2	3	4	5	6	7	8	9	10	11	12	13
o	p	q	r	s	t	u	v	w	x	y	z	0	1
14	15	16	17	18	19	20	21	22	23	24	25	26	27
2	3	4	5	6	7	8	9	,	.	(	)	-	’
28	29	30	31	32	33	34	35	36	37	38	39	40	41

Znak „spacji” posiada kod 42, kody od 46 do 60 są zarezerwowane, kody od 61 do 63 są wolne

Kod PF1

Dodatkowo wprowadzamy następujące sekwencje sterujące:

- `ESC1` (kod 43) służącą do zamiany litery małej występującej zaraz za sekwencją na dużą.
- `ESC2` (kod 44) służącą do uzyskania znaków diakrytycznych. Sekwencja
  - `ESC2 , litera` dodaje „ogonek” do litery,
  - `ESC2 . litera` dodaje „kropkę” do litery,
  - `ESC2 - litera` dodaje „przekreślenie” do litery,
  - `ESC2 ' litera` dodaje „kreskę nad” do litery.
- `NL` (kod 45) powodujący przejście do nowego wiersza.
- Kod `ESC1` zawsze występuje przed `ESC2`.

Kod PF1

Jak widać w kodzie PF1 największa wartość używanego kodu wynosi 63. Stąd wniosek, że musimy używać co najmniej 6 bitów do zapisania kodów. Spróbujmy zakodować następujące zdanie:

Żółty.

Kod PF1

Zdanie

Żółty.

zapiszemy przy pomocy następującej sekwencji kodów:

43, 44, 37, 25	Ż	101011, 101100, 100101, 011001
44, 36, 14	ó	101100, 100100, 001110
44, 40, 11	ł	101100, 101000, 001011
19	t	010011
24	y	000000
37	.	101010

Kod PF1

Kody dziesiętnie:
43, 44, 37, 25, 44, 36, 14, 44, 40, 11, 19, 24, 37

Kody binarnie na 6 bitach:
101011, 101100, 100101, 011001, 101100, 100100, 001110,
101100, 101000, 001011, 010011, 011000, 100101

Kody 6 bitowe sklejone w jedne ciąg 0-1:
101011101100100101011001101100100100001110
101100101000001011010011011000101010

Ciąg 0-1 "pocięty" na bajty ("paczki" po 0 bitów):
10101110, 11001001, 01011001, 10110010, 01000011,
10101100, 10100000, 10110100, 11011000, 100101??

Bajty:
174, 201, 185, 178, 67,
172, 160, 180, 216, 148??

Kod PF1

```
Bajty:
174, 201, 185, 178, 67, 172, 160, 180, 216, 148

zapisane szesnastkowo jako
AE, C9, 59, B2, 43,
AC, A0, B4, D8, 94

zapisujemy do pliku zolty.pf1

$ echo -n -e '\xae\xc9\x59\xb3\x43\xac\xa0\xb4\xd8\x94' > zolty.pf1
$ cat zolty.pf1
??Y?C?????$ od -t x1 zolty.pf1
0000000 ae c9 59 b3 43 ac a0 b4 d8 94
0000012
$ wc zolty.pf1
0  1 10 zolty.pf1
```

Kod PF1

Nasza zabawa miała dwa cele.

- Ukazanie porzeby istnienia znaków sterujących jako narzędzia niezbędnego w sytuacji, gdy trzeba zakodować więcej znaków niż dotępna przestrzeń kodowa.
- Zobrazowanie sposobu postępowania, gdy bitowo zapisane kody nie mają długości będącej wielokrotnością liczby 8.

ASCII

ASCII American Standard Code for Information Interchange. W kodowaniu tym określono kody dla

- małych (97-122) i dużych (65-90) liter alfabetu łaciński;
- cyfr (48-57);
- pewnej grupy znaków jak np. `(`, `:`, + itp. (32-47, 58-64, 91-96, 123-126);
- niedrukowalnych znaków sterujących przepływem danych, np. `ACK` – potwierdzenie, czy `BEL` – sygnał dźwiękowy (0-31 oraz 127).

Zakres kodów ASCII rozciąg się od 0 do 127, czyli wymaga wykorzystania co najmniej 7 bitów. Ponieważ większość komputerów była 8-bitowa (czyli posługująca się informacjami dzielonymi na kawałki po 8 bitów) więc pozostawało jeszcze 128 wolnych miejsc o numerach od 128 do 255.

Znaki ASCII nie pokrywały zapotrzebowania narodowości posługujących się literami alfabetu łacińskiego ze specyficznymi znakami diakrytycznymi (Niemcy, Polska) lub wręcz zupełnie niestandardowymi znakami (Grecja, Rosja), nie mówiąc już o fizycznej niemożności reprezentacji liczniejszych zbiorów znaków (Chiny).

Ze względu na powstałe zapotrzebowanie, do reprezentacji znaków narodowych wykorzystano wolne 128 pozycji.

Zakres kodów ASCII rozciąg się od 0 do 127, czyli wymaga wykorzystania co najmniej 7 bitów. Ponieważ większość komputerów była 8-bitowa (czyli posługująca się informacjami dzielonymi na kawałki po 8 bitów) więc pozostawało jeszcze 128 wolnych miejsc o numerach od 128 do 255.

Znaki ASCII nie pokrywały zapotrzebowania narodowości posługujących się literami alfabetu łacińskiego ze specyficznymi znakami diakrytycznymi (Niemcy, Polska) lub wręcz zupełnie niestandardowymi znakami (Grecja, Rosja), nie mówiąc już o fizycznej niemożności reprezentacji liczniejszych zbiorów znaków (Chiny).

Ze względu na powstałe zapotrzebowanie, do reprezentacji znaków narodowych wykorzystano wolne 128 pozycji.

Zakres kodów ASCII rozciąg się od 0 do 127, czyli wymaga wykorzystania co najmniej 7 bitów. Ponieważ większość komputerów była 8-bitowa (czyli posługująca się informacjami dzielonymi na kawałki po 8 bitów) więc pozostawało jeszcze 128 wolnych miejsc o numerach od 128 do 255.

Znaki ASCII nie pokrywały zapotrzebowania narodowości posługujących się literami alfabetu łacińskiego ze specyficznymi znakami diakrytycznymi (Niemcy, Polska) lub wręcz zupełnie niestandardowymi znakami (Grecja, Rosja), nie mówiąc już o fizycznej niemożności reprezentacji liczniejszych zbiorów znaków (Chiny).

Ze względu na powstałe zapotrzebowanie, do reprezentacji znaków narodowych wykorzystano wolne 128 pozycji.

Szkoda tylko, że każda narodowość zrobiła to niezależnie od innych.

W ten oto sposób powstały **strony kodowe**, czyli zestawy 255 znaków o wspólnej pierwszej połowie, natomiast różniące się zasadniczo w drugiej.

Dlatego manipulując jakimkolwiek tekstem, jeśli chcemy poprawnie odczytać niestandardowe znaki alfabetu łacińskiego, MUSIMY wiedzieć przy pomocy jakiej strony kodowej został on zapisany.

Szkoda tylko, że każda narodowość zrobiła to niezależnie od innych.

W ten oto sposób powstały **strony kodowe**, czyli zestawy 255 znaków o wspólnej pierwszej połowie, natomiast różniące się zasadniczo w drugiej.

Dlatego manipulując jakimkolwiek tekstem, jeśli chcemy poprawnie odczytać niestandardowe znaki alfabetu łacińskiego, MUSIMY wiedzieć przy pomocy jakiej strony kodowej został on zapisany.

Szkoda tylko, że każda narodowość zrobiła to niezależnie od innych.

W ten oto sposób powstały **strony kodowe**, czyli zestawy 255 znaków o wspólnej pierwszej połowie, natomiast różniące się zasadniczo w drugiej.

Dlatego manipulując jakimkolwiek tekstem, jeśli chcemy poprawnie odczytać niestandardowe znaki alfabetu łacińskiego, MUSIMY wiedzieć przy pomocy jakiej strony kodowej został on zapisany.

Standard	ą	ć	ę	ł	ń	ó	ś	ż	ź
ISO 8859-2	B1	E6	EA	B3	F1	F3	B6	BF	BC
CP 1250	B9	E6	EA	B3	F1	F3	9C	BF	9F
Mazowia	86	8D	91	92	A4	A2	9E	A7	A6
Unicode	105	107	119	142	144	F3	15B	17C	17A

- ISO 8859-2, nazywane także latin2, jest kodowaniem charakterystycznym dla systemów rodziny UNIX-owych.
- CP 1250, nazywane także win-1250, jest kodowaniem charakterystycznym dla systemów rodziny Windows.
- Mazowia –kodowanie opracowane na potrzeby polskiego komputera Mazovia.
- Unicode – o tym dalej.

Problemy

- Oczywisty – wiele różnych stron kodowych nawet dla tego samego języka.
- Trudności z obsługą tekstów wielojęzycznych.
- Zbyt mała przestrzeń dla kodów niektórych języków, np. chiński.

Unicode – najważniejsze cechy

Jednoznaczność. Jeden kod odpowiada jednemu znakowi i odwrotnie.

Uniwersalność. Wszystkie powszechnie używane języki oraz symbole.

Efektywność. Identyfikacja znaku nie zależy od sekwencji sterującej czy znaków następujących bądź poprzedzających.

Identyfikacja nie reprezentacja. Znak a nie jego wygląd.

Znaczenie. Własności znaków (np. kolejność alfabetyczna) nie zależą od położenia w tabeli kodów ale są określone w tablicy własności.

Czysty tekst.

Logiczny porządek.

Ujednolicenie. Identyczne znaki o różnym znaczeniu zastąpiono jednym.

Kodowanie UTF

Unicode i UTF (ang. *UCS Transformation Format*, UCS – ang. *Universal Character Set*) to ściśle powiązane ze sobą, ale jednak różne, pojęcia. Pierwsze z nich wiąże się, jak widzieliśmy, z przyporządkowaniem odpowiednich kodów konkretnym znakom. UTF określa natomiast jak owe liczby zostaną przedstawione przy pomocy zer i jedynek. Mimo, że w praktyce nie jest konieczna dogłębna znajomość poszczególnych sposobów kodowania, to już mylenie Unicode i UTF może powodować sporo problemów praktycznych.

Kodowanie UTF-32

Kodowanie UTF-32 jest najprostszym z kodowań, gdyż w odróżnieniu od UTF-16 i UTF-8 ma stałą długość każdego znaku wynoszącą 32 bity (tj. 4 bajty). Wadą takiego rozwiązania jest częste marnowanie dużej ilości pamięci. Dzieje się tak, gdyż 4 bajty mogą przyjąć wartości od `0x00000000` do `0xFFFFFFFF`, natomiast wszystkie znaki europejskich języków mieszczą się w przedziale od `0x0000` do `0xFFFF`. Innymi słowy, kodując w UTF-32 prawie zawsze przynajmniej połowa bitów jest zerowa. Stała ilość bajtów ma też swoje zalety — pozwala na dokładne odnalezienie *n*-tego znaku w tekście, bez konieczności analizowania znaków poprzedzających.

Kodowanie UTF-32

Ponieważ znak kodowany jest na 4 bajtach, więc w naturalny sposób powstaje problem zapisu: big endian czy little endian? Standardowo stosowany jest zapisy big endian. Istnieje jednak sposób, aby wprost określić stosowanie BE lub LE. Ten sam sposób jest często używany, żeby w ogóle zaznaczyć, że tekst jest zakodowany w UTF. W tym celu wykorzystuje się specjalny ciąg znaków podawany na samym początku tekstu, tzw. BOM (ang. *Byte Order Mark*).

Kodowanie UTF-32

Kodowanie	BOM dla LE	BOM dla BE
UTF-32	0xff 0xfe 0x00 0x00	0x00 0x00 0xfe 0xff
UTF-16	0xff 0xfe	0xfe 0xff

The UTF-8 representation of the BOM is the byte sequence 0xEF,0xBB,0xBF. The Unicode Standard permits the BOM in UTF-8, but does not require or recommend its use. Byte order has no meaning in UTF-8, so its only use in UTF-8 is to signal at the start that the text stream is encoded in UTF-8. Note that some recipients of UTF-8 encoded data do not expect a BOM. Where UTF-8 is used transparently in 8-bit environments, the use of a BOM will interfere with any protocol or file format that expects specific ASCII characters at the beginning, such as the use of `#!` of at the beginning of Unix shell scripts.

Kodowanie UTF-8

Przeciwieństwem kodowania UTF-32 jest kodowanie UTF-8. Znaki nie mają w nim stałej długości. Mogą składać się z 1 do 6 bajtów. Pierwsze znaki Unicode są kodowane jednym bajtem i tym samym są identyczne z kodem ASCII. Kolejne znaki są odpowiednio kodowane 2, 3, 4, 5 i 6 bajtami. Zasada kodowania w UTF-8 jest następująca. Pierwsze bity, pierwszego bajtu określają z ilu bajtów składa się znak. Kolejne bajty mają za to stały 2-bitowy prefiks.

Kodowanie UTF-8

Bits of code point	Range	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	B
7	0000-007F	0xxxxxxx					
11	0080-07FF	110xxxxx	10xxxxxx				
16	0800-FFFF	1110xxxx	10xxxxxx	10xxxxxx			
21	10000-1FFFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx		
26	200000-3FFFFFF	111110xx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	
31	4000000-7FFFFFFF	1111110x	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	10x

Uwagi

- Ponieważ budowa kodów UTF-8 pozwala zapisać ten sam znak na kilka różnych sposobów, więc standard dookreśla, że poprawnym zapisem jest tylko najkrótszy z możliwych.

Uwagi

- Znając strukturę kodów UTF-8 jasnym staje się, dlaczego to kodowanie pominięte zostało w tabeli BOM. Otóż w kodowaniu zorientowanym bajtowo odpowiednią decyzję można podjąć odczytując pojedyńczy (kolejny) bajt i analizując jego bity.

Kodowanie UTF-8

Byte oriented coding

UTF-8 can contain a BOM. However, it makes no difference as to the endianness of the byte stream. UTF-8 always has the same byte order. An initial BOM is only used as a signature – an indication that an otherwise unmarked text file is in UTF-8.
UTF-8 is byte oriented, so there's not an issue regarding endianness. The smallest unit is a byte (thus it is byte-oriented) so the alogrithm reads or writes one byte at a time. A byte is represented the same way on all machines – the first byte is always the first byte, the second byte is always the second byte etc. regardless of endianness.
Byte oriented means that we read a byte at a time, and decide what to do based on that byte. In UTF-8, we decide what to do with a byte based on its high-order bits. For UTF-16, the smallest unit is a 16-bit word, and for UTF-32, the smallest unit is a 32-bit word, so they are not byte-oriented. Both in UTF-16 and UTF-32 we deal with multiple bytes at a time, and have to organize them into words so in that cases endianness matters. The algorithm reads or writes one word at a time (2 bytes, or 4 bytes). The order of the bytes in each word is different on big-endian and little-endian machines.

Kodowanie UTF-8

Example

Let us consider how to encode the Euro sign.

1. The Unicode code point for Euro sign is U+20ac.
2. According to the scheme table above, this will take three bytes to encode, since it is between U+0800 and U+ffff.
3. Hexadecimal 20ac is binary 00100000 10101100. The two leading zeros are added because, as the scheme table shows, a three-byte encoding needs exactly sixteen bits from the code point.
4. Because it is a three-byte encoding, the leading byte starts with three 1s, then a 0 (1110...)
5. The remaining bits of this byte are taken from the code point (11100010), leaving ...000010101100.
6. Each of the continuation bytes starts with 10 and takes six bits of the code point (so 10000010, then 10101100).

The three bytes 11100010 10000010 10101100 can be more concisely written in hexadecimal, as e2 82 ac.

Kodowanie UTF-8

Example

Let us consider how to encode the Euro sign.

1. The Unicode code point for Euro sign is U+20ac.
2. According to the scheme table above, this will take three bytes to encode, since it is between U+0800 and U+ffff.
3. Hexadecimal 20ac is binary 00100000 10101100. The two leading zeros are added because, as the scheme table shows, a three-byte encoding needs exactly sixteen bits from the code point.
4. Because it is a three-byte encoding, the leading byte starts with three 1s, then a 0 (1110...)
5. The remaining bits of this byte are taken from the code point (11100010), leaving ...000010101100.
6. Each of the continuation bytes starts with 10 and takes six bits of the code point (so 10000010, then 10101100).

The three bytes 11100010 10000010 10101100 can be more concisely written in hexadecimal, as e2 82 ac.

Kodowanie UTF-8

Example

Let us consider how to encode the Euro sign.

1. The Unicode code point for Euro sign is U+20ac.
2. According to the scheme table above, this will take three bytes to encode, since it is between U+0800 and U+ffff.
3. Hexadecimal 20ac is binary 00100000 10101100. The two leading zeros are added because, as the scheme table shows, a three-byte encoding needs exactly sixteen bits from the code point.
4. Because it is a three-byte encoding, the leading byte starts with three 1s, then a 0 (1110...)
5. The remaining bits of this byte are taken from the code point (11100010), leaving ...0000010101100.
6. Each of the continuation bytes starts with 10 and takes six bits of the code point (so 10000010, then 10101100).

The three bytes 11100010 10000010 10101100 can be more concisely written in hexadecimal, as e2 82 ac.

Kodowanie UTF-8

Example

Let us consider how to encode the Euro sign.

1. The Unicode code point for Euro sign is U+20ac.
2. According to the scheme table above, this will take three bytes to encode, since it is between U+0800 and U+ffff.
3. Hexadecimal 20ac is binary 00100000 10101100. The two leading zeros are added because, as the scheme table shows, a three-byte encoding needs exactly sixteen bits from the code point.
4. Because it is a three-byte encoding, the leading byte starts with three 1s, then a 0 (1110...)
5. The remaining bits of this byte are taken from the code point (11100010), leaving ...000010101100.
6. Each of the continuation bytes starts with 10 and takes six bits of the code point (so 10000010, then 10101100).

The three bytes 11100010 10000010 10101100 can be more concisely written in hexadecimal, as e2 82 ac.

Kodowanie UTF-8

Example

Let us consider how to encode the Euro sign.

1. The Unicode code point for Euro sign is U+20ac.
2. According to the scheme table above, this will take three bytes to encode, since it is between U+0800 and U+ffff.
3. Hexadecimal 20ac is binary 00100000 10101100. The two leading zeros are added because, as the scheme table shows, a three-byte encoding needs exactly sixteen bits from the code point.
4. Because it is a three-byte encoding, the leading byte starts with three 1s, then a 0 (1110...)
5. The remaining bits of this byte are taken from the code point (11100010), leaving ...000010101100.
6. Each of the continuation bytes starts with 10 and takes six bits of the code point (so 10000010, then 10101100).

The three bytes 11100010 10000010 10101100 can be more concisely written in hexadecimal, as e2 82 ac.

Kodowanie UTF-8

Example

Let us consider how to encode the Euro sign.

1. The Unicode code point for Euro sign is U+20ac.
2. According to the scheme table above, this will take three bytes to encode, since it is between U+0800 and U+ffff.
3. Hexadecimal 20ac is binary 00100000 10101100. The two leading zeros are added because, as the scheme table shows, a three-byte encoding needs exactly sixteen bits from the code point.
4. Because it is a three-byte encoding, the leading byte starts with three 1s, then a 0 (1110...)
5. The remaining bits of this byte are taken from the code point (11100010), leaving ...000010101100.
6. Each of the continuation bytes starts with 10 and takes six bits of the code point (so 10000010, then 10101100).

The three bytes 11100010 10000010 10101100 can be more concisely written in hexadecimal, as e2 82 ac.

Kodowanie UTF-8

Example

Let us decode sign (letter) encoded in UTF-8.

1. Bytes used to encode this sign (letter) are: c4 84

```
$ echo -n -e '\xc4\x84' > unknown_utf8.txt
$ od -t x1 unknown_utf8.txt
0000000 c4 84
0000002
```

2. Hexadecimal c4 84 is a binary 11000100 10000100
3. From UTF-8 coding table we can divide this sequence into the following sections: 11000100 10000100.
4. 11 bits are used to encode Unicode code 001 00000100 can be written in hexadecimal, as 01 04 and in Unicode form as U+0104.
5. Checking the sign (letter) encoded as U+0104 on https://unicode-table.com/en/ we can see that this is an Ą letter called *Latin Capital Letter a with Ogonek*.

```
$ cat unknown_utf8.txt
Ą
```

Kodowanie UTF-8

Example

Let us decode sign (letter) encoded in UTF-8.

1. Bytes used to encode this sign (letter) are: c4 84

```
$ echo -n -e '\xc4\x84' > unknown_utf8.txt
$ od -t x1 unknown_utf8.txt
0000000 c4 84
0000002
```

2. Hexadecimal c4 84 is a binary 11000100 10000100
3. From UTF-8 coding table we can divide this sequence into the following sections: 110**00100** 10**000100**.
4. 11 bits are used to encode Unicode code 001 00000100 can be written in hexadecimal, as 01 04 and in Unicode form as U+0104.
5. Checking the sign (letter) encoded as U+0104 on `https://unicode-table.com/en/` we can see that this is an Ą letter called *Latin Capital Letter a with Ogonek*.

```
$ cat unknown_utf8.txt
Ą
```

Kodowanie UTF-8

Example

Let us decode sign (letter) encoded in UTF-8.

1. Bytes used to encode this sign (letter) are: c4 84

```
$ echo -n -e '\xc4\x84' > unknown_utf8.txt
$ od -t x1 unknown_utf8.txt
0000000 c4 84
0000002
```

2. Hexadecimal c4 84 is a binary 11000100 10000100
3. From UTF-8 coding table we can divide this sequence into the following sections: 110**00100** 10**000100**.
4. 11 bits are used to encode Unicode code 001 00000100 can be written in hexadecimal, as 01 04 and in Unicode form as U+0104.
5. Checking the sign (letter) encoded as U+0104 on https://unicode-table.com/en/ we can see that this is an Ą letter called *Latin Capital Letter a with Ogonek*.

```
$ cat unknown_utf8.txt
Ą
```

Kodowanie UTF-8

Example

Let us decode sign (letter) encoded in UTF-8.

1. Bytes used to encode this sign (letter) are: c4 84

```
$ echo -n -e '\xc4\x84' > unknown_utf8.txt
$ od -t x1 unknown_utf8.txt
0000000 c4 84
0000002
```

2. Hexadecimal c4 84 is a binary 11000100 10000100
3. From UTF-8 coding table we can divide this sequence into the following sections: 110**00100** 10**000100**.
4. 11 bits are used to encode Unicode code 001 00000100 can be written in hexadecimal, as 01 04 and in Unicode form as U+0104.
5. Checking the sign (letter) encoded as U+0104 on https://unicode-table.com/en/ we can see that this is an *Ą* letter called *Latin Capital Letter a with Ogonek*.

```
$ cat unknown_utf8.txt
Ą
```

Kodowanie UTF-8

Example

Let us decode sign (letter) encoded in UTF-8.

1. Bytes used to encode this sign (letter) are: c4 84

```
$ echo -n -e '\xc4\x84' > unknown_utf8.txt
$ od -t x1 unknown_utf8.txt
0000000 c4 84
0000002
```

2. Hexadecimal c4 84 is a binary 11000100 10000100
3. From UTF-8 coding table we can divide this sequence into the following sections: 110**00100** 10**000100**.
4. 11 bits are used to encode Unicode code 001 00000100 can be written in hexadecimal, as 01 04 and in Unicode form as U+0104.
5. Checking the sign (letter) encoded as U+0104 on https://unicode-table.com/en/ we can see that this is an Ą letter called *Latin Capital Letter a with Ogonek*.

```
$ cat unknown_utf8.txt
Ą
```

Graficzny plik typu bmp

Pakiet protokołół TCP/IP