Zmatky při používání češtiny a možná řešení

doc. Ing. Pavel Herout, Ph.D.

Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Kódování znaků je vztah tří (čtyř) součástí

abstrakce

papír

0 ≈ A
1 ≈ B
2 ≈ C

přirozená čísla

znaková sada

Coded Character Set

v počítači je vše číslo

realita

paměť počítače

00 00 ≈ A
00 01 ≈ B
00 02 ≈ C

bajty nebo násobky

Character Encoding Form

font

na pozici 0 je

„obrázek“ 

tvar znaku (glyph)

realita

soubor na disku

00 ≈ A
01 ≈ B
02 ≈ C

bajty

kódovací schéma, kódování

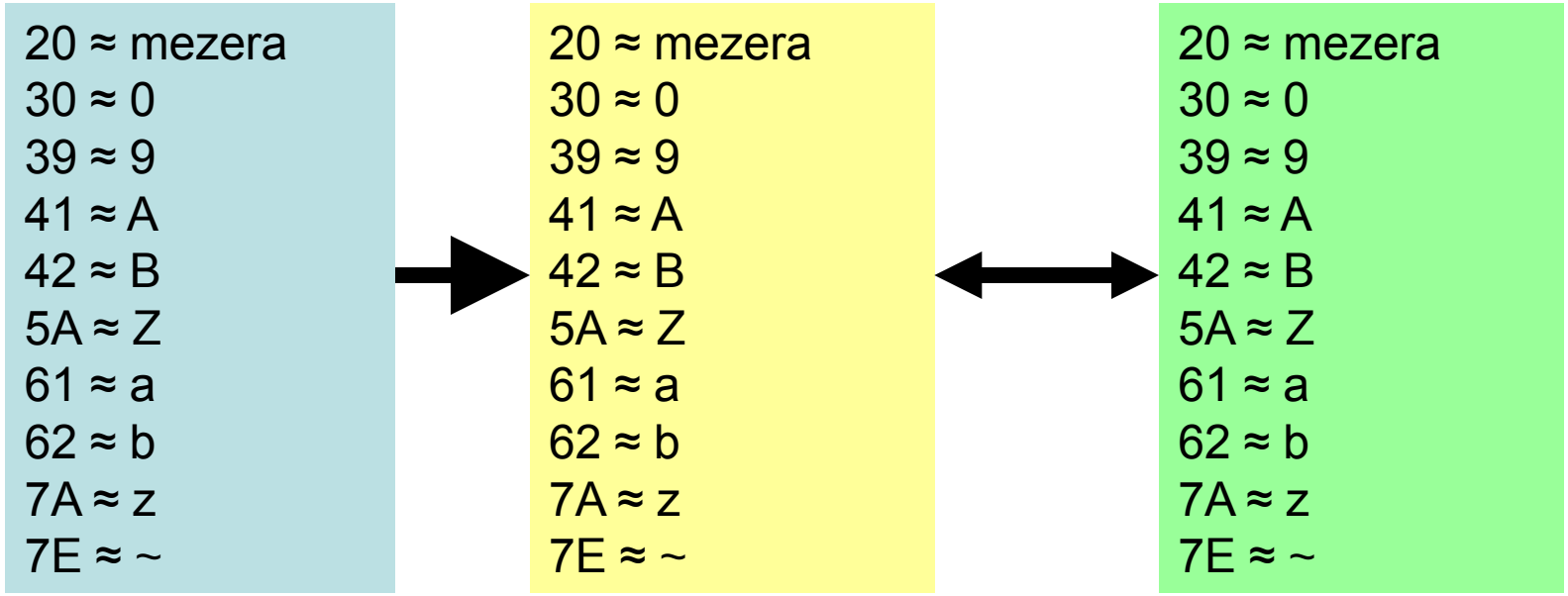
Character Encoding Scheme

charset

Historie – 50. léta – 8bitové počítače v USA

ASCII – „všechny znaky z anglické klávesnice“, tj. velká a malá neakcentovaná písmena latinky, číslice, interpunkci a speciální a řídicí znaky

sedmibitový kód – k dispozici 128 znaků, z toho 32 řídicích (dálnopis)



z 32 řídicích znaků se dnes běžně využívají

- 09 ≈ tabulátor
- 0A ≈ <LF>
- 0D ≈ <CR>

Historie – 60. až 80. léta – 8 až 16bitové počítače mimo USA

požadavky na znaky národních abeced – akcentované znaky

rozšíření ASCII o zbývajících 128 znaků do 8 bitového rozsahu
128 znaků nestačí pro všechny požadavky (jen čeština potřebuje 30 akcentovaných znaků – áčďéěíňóřšťúůýž)

znakové sady (abstraktní vrstvu) vytvářeli naprosto nekoordinovaně
- mezinárodní standardizační instituce – ISO-8859-1 až ISO-8859-15
- národní standardizační instituce – KOI-8 ČS2
- firmy – v návaznosti na konkrétní systém – CP1250, AppleCE, IBM852
- jednotlivci – Kameničtí KEYBCS2

- připravit kvalitní abstraktní vrstvu je problém – mnoho hledisek, např. řazení, vtahy mezi malými a velkými písmeny atd.

chaos nebyl problém !
- počítače byly izolované – minimální výměna informací
    - není e-mail ani WWW (velké disky, děrné pásky a štítky, diskety)
- akcenty byly záležitostí pouze textových procesorů, které rozpoznaly „svoje“ kódování
- vícejazyčné texty (např. česko-francouzský) naprosto výjimečné

Historie – 60. až 80. léta – 8 až 16bitové počítače mimo USA

osmibitový kód **nedělá problémy** na disku

nedělá problémy ani v paměti

- 8bitový počítač – žádná úprava
- 16bitové počítač – vyšší bajt (pokud je použit) je vždy nulový

41 ≈ A
42 ≈ B
5A ≈ Z
7E ≈ ~

→

00 41 ≈ A
00 42 ≈ B
00 5A ≈ Z
00 7E ≈ ~

↔

41 ≈ A
42 ≈ B
5A ≈ Z
7E ≈ ~

znaková sada a kódování se jmenují stejně

Problémy – ukázka rozdílů v abstraktní vrstvě

ISO-8859-2	windows-1250	IBM852	MacCentralEurope
C1 ≈ Á	C1 ≈ Á	B5 ≈ Á	E7 ≈ Á
C8 ≈ Č	C8 ≈ Č	AC ≈ Č	89 ≈ Č
CF ≈ Ď	CF ≈ Ď	D2 ≈ Ď	91 ≈ Ď
A9 ≈ Š	8A ≈ Š	E6 ≈ Š	E1 ≈ Š
AB ≈ Ť	8D ≈ Ť	9B ≈ Ť	E8 ≈ Ť
AE ≈ Ž	8E ≈ Ž	A6 ≈ Ž	EB ≈ Ž

velmi nebezpečná je podobnost ISO-8859-2 a windows-1250
- liší se jen v 6 znacích š, ť, ž, Š, Ť, Ž
- chybně rozpoznané kódování se dá lehce přehlédnout

kódování IBM852 se používá na příkazové řádce
- program, který má kódování správně v okně, jej nemá správně na příkazové řádce

existují stovky osmibitových kódování, pro češtinu je jich významných asi 10

Problém – dokument si v sobě nenese informaci o použitém kódování

tato informace se (pokud vůbec) předává jinou cestou

některé jazyky pro popis dat mají možnost uložit informaci o použitém kódování

HTML

```
<html><head>
 <meta http-equiv="Content-Type" content="text/html;
                                          charset=ISO-8859-2">
```

XML

```
<?xml version="1.0" encoding="ISO-8859-2"?>
```

MIME – Multipurpose Internet Mail Extensions

```
Content-Type: text/plain; charset="ISO-8859-2"
```

problém je, když se hlavičky těchto souborů bezmyšlenkovitě opisují a skutečné kódování je jiné

„Pokud předáváte obecný dokument v osmibitovém kódování, měl by být v ISO-8859-2.“

- toto je akademická rada, prakticky neuskutečnitelná, máme-li již dokument a nemáme vhodný konverzní program
- u některých editorů lze kódování výstupního dokumentu přednastavit
- soubory jednoho projektu by měly mít stejné kódování

Problém – různá synonyma osmibitových kódování

totéž kódování má různá pojmenování a to i v rámci produktů jedné firmy
• nová jména vymýšlejí uživatelé, nikoliv tvůrci
• rozlišuje se hlavní **kanonické jméno** (registrované v IANA) a ostatní jména – **aliasy**

US-ASCII– American Standard Code for Information Interchange

• ISO646-US, IBM367, ASCII, cp367, default, ascii7, ANSI_X3.4-1986, iso-ir-6, us, 646, iso_646.irv:1983, csASCII, ANSI_X3.4-1968, ISO_646.irv:1991

ISO-8859-2 – Latin Alphabet No. 2

• ibm912, l2, ibm-912, cp912, ISO_8859-2:1987, ISO_8859-2, latin2, csISOLatin2, iso8859_2, 912, 8859_2, ISO8859-2, iso-ir-101

windows-1250 – Windows Eastern European

• cp1250, CP1250, cp5346

IBM852 – MS-DOS Latin-2

• 852, ibm-852, csPCp852, cp852, ibm852

některé aplikace označují osmibitové charsety jako ANSI

další „lidová tvořivost“ existuje ve jmenných pojmenováních, např. Středoevropské jazyky (Windows)

to znamená, že i když kódování známe, nemusíme dokument správně zobrazit, protože neznáme alias v příslušném programu (strojově)

Současnost – znaková sada s více než 256 znaky

chaos, který nevadil do konce 80. let, začíná s rozvojem Internetu výrazně vadit

je požadavek na jednotnou znakovou sadu s více než 256 znaky

bohužel tuto sadu začínají připravovat dvě různé instituce

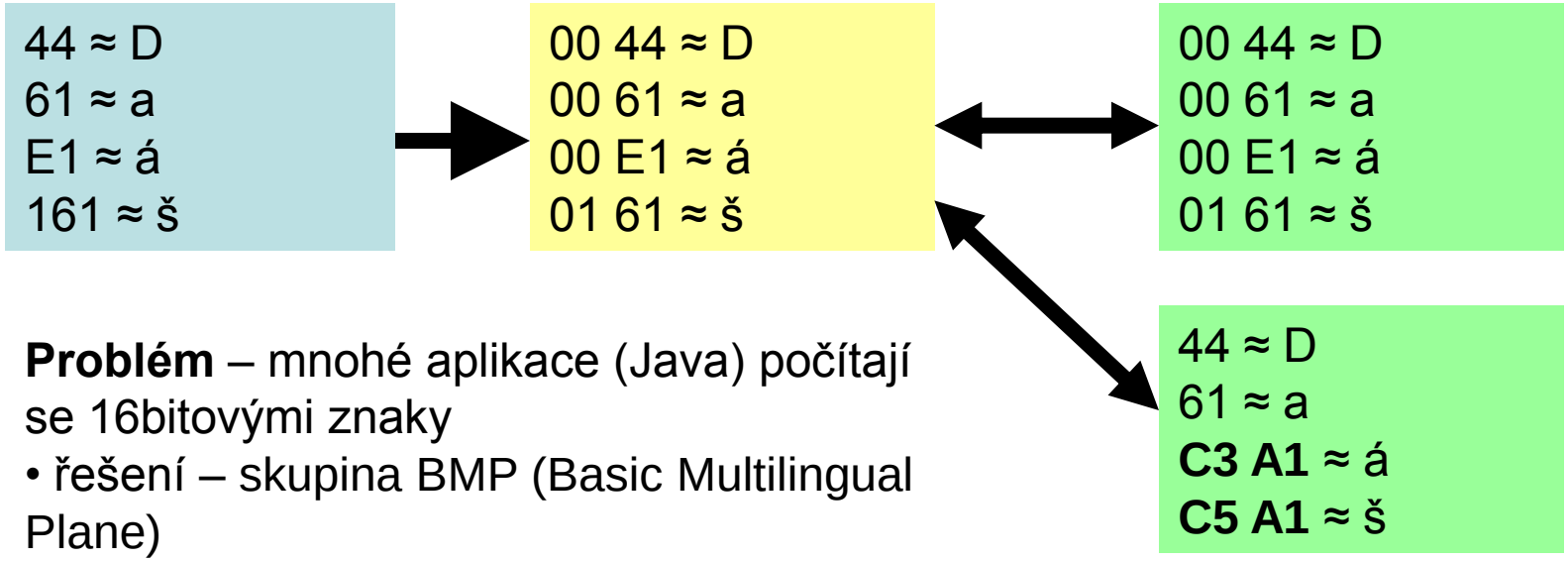
- **ISO** (International Organization for Standardization)
  - od 1989
  - znaková sada ISO/IEC 10646
  - ve zkratce UCS (Universal Character Set)
- **Unicode** (Unicode Consortium)
  - od 1990
  - znaková sada Unicode
  - ve zkratce Unicode
- naštěstí se obě organizace domlouvají hned na počátku prací a od roku 1991 pracují na sjednocení obou znakových sad, což bylo dokončeno v 1993
- z běžného pohledu nemá význam obě sady rozlišovat a běžně se považují za synonyma a používá se společné označení Unicode

Problém – pouze v duplicitním pojmenování, kdy se občas vyskytne UCS

Unicode

prvních 127 znaků je totožných s ASCII – důležité z hlediska historie

původní Unicode bylo plánováno na 65535 znaků (tj. 16 bitů v paměti)
- to přestalo stačit v roce 2003 (od roku 1996 se s tím počítalo)
- dnes je registrováno přes 100 tisíc znaků



Problém – mnohé aplikace (Java) počítají se 16bitovými znaky
- řešení – skupina BMP (Basic Multilingual Plane)
- BMP obsahuje všechny znaky používané v Evropě a Americe plus základní ideografická písma čínštiny, japonštiny a korejštiny (HAN písmo)
- zástupné páry (surrogate pairs) řeší znaky nad 16 bitů

označování znaků U+01A5

Problémy při zápisu na disk

máme jednu znakovou sadu, ale protože znak přesahuje rozsah jednoho bajtu, používá se více způsobů zápisů na disk (kódovacích schémat)

Problém pořadí bajtů

ukládáme-li do paměti (do souboru) vícebajtové entity, je třeba rozlišovat pořadí bajtů

A = 0041

• little-endian LE (vyšší řády na vyšší adrese) 41 00

• big-endian BE (vyšší řády na nižší adrese) 00 41

toto platí i pro např. čtyřbajtové entity 00 00 00 41 či 41 00 00 00
• způsob ukládání little- nebo big-endian závisí na platformě (Windows LE), procesoru (Intel LE, Motorola BE), programovacím jazyce (Java vždy BE), aplikaci, ...

proč to potřebujeme?
• většina současných souborových systémů pracuje s bajty – pro správnou serializaci znaků do bajtového proudu

Problém kódovacích schémat

UCS-2, UTF-16BE

00 44 ≈ D
00 61 ≈ a
00 E1 ≈ á
01 61 ≈ š

00 44 ≈ D
00 61 ≈ a
00 E1 ≈ á
01 61 ≈ š

UTF-8

44 ≈ D
61 ≈ a
C3 A1 ≈ á
C5 A1 ≈ š

UTF-16LE

44 00 ≈ D
61 00 ≈ a
E1 00 ≈ á
61 01 ≈ š

tři základní UTF (Unicode Transformation Format)

UCS-2 je kódování z ISO 10646

další kódování (méně používaná)

- UTF-16
- UTF-32
- UTF-32BE
- UTF-32LE
- UCS-4

Problém značky bajtového pořadí

využívá se pro identifikaci pořadí bajtů – BE nebo LE

initial byte order mark (BOM) – je umístěna na samém začátku souboru

pro tento účel definuje Unicode dva kódové body

- U+FEFF = ZERO WIDTH NO-BREAK SPACE (byte order mark) (pevná mezera nulové délky)
- U+FFFE = not a character code

pro UTF-16 pro BE má tvar `FE FF` a pro LE pak `FF FE`

je-li BOM načtena správně, pak se pevná mezera nulové délky ze své podstaty nemůže zobrazit

používá se u UTF-16 (ne u UTF-16BE nebo UTF-16LE)

může se používat u UTF-8

BOM má nyní významnou (vedlejší) funkci – určení použitého kódování

UTF-8

bylo vytvořeno proto, aby se znaky Unicode daly zakódovat posloupností bajtů, protože s bajty umí pracovat každá aplikace a každý souborový systém

je to obecně rozšířený a přijímaný formát – často se UTF-8 pokládá za Unicode

výhody
- pro texty využívající jen znaky anglické abecedy je UTF-8 totožná s US-ASCII
  - využívá se jen jeden bajt na jeden znak
  - s US-ASCII umí pracovat každá aplikace
- pro akcentované znaky se využívají dva bajty

44 ≈ D
61 ≈ a
C3 A1 ≈ á
C5 A1 ≈ š

základní nevýhoda UTF-8 – znaky nemají stejnou délku, tzn. není možné skočit přímo na určitý znak („přeskoč prvních 20 znaků“)
- pravděpodobnost „omylu“ (považování poloviny znaku za celý znak) je omezena principem kódovacího schématu

UTF-8 je kódování, které se nejsnáze rozpozná – zobrazíme dokument běžným editorem a pokud místo každého akcentovaného znaku vidíme dva nesmyslné znaky, je to UTF-8

UTF-8 a BOM

BOM není nezbytný – určovat bajtové pořadí zde nemá smysl

v současné době je mnoha aplikacemi BOM v UTF-8 používán jako označení formátu dokumentu

BOM má tvar `EF BB BF` opět na samém začátku souboru

tyto tři bajty jsou editorem/prohlížečem použity k detekci kódování dokumentu a nezobrazují se

Problém – některé aplikace mají s BOM v UTF-8 potíž

• při neznalosti principu je těžké problém odhalit, protože při otevření souboru BOM „neexistuje“

• odstranění BOM je obtížné – SciTE

Zápis akcentů pomocí US-ASCII

- princip – akcentované znaky se zapíší opisem pomocí neakcentovaných
- dva základní způsoby
  - slovní popis `&aacute;`
  - číslo kódového bodu `&#353;` nebo `&#x161;` (většinou Unicode)
    nebo `=E1 (e-mail)`
    nebo `%E1 (URL)` (většinou ISO-8859-2)
- možné jsou i další způsoby – BASE 64