Caractères

Représentation

Evolution des codages

Unicode

Représentation

La représentation des caractèes est une convention

- associe à chaque caractère de l'alphabet un nombre binaire arbitraire

Il n'y a pas de représentation canonique

- il existe un ordre sur certains caractères 'a' < 'b' < 'c'
- mais pas sur tous ';' < '+' ?

Le codage devrait respecter l'ordre alphabétique quand il existe

- code('a') < code('b') < code('c')

pour simplifier les traitements

Opérations sur les caractères :

- comparaison (=), ordre lexicographique (<), transcodage

Evoluation du codage

Nécessité de définir un standard pour l'échange de données entre machines

Codage défini par

- nb. bits pour représenter un caractère
- alphabet pris en compte
- codes assignés aux caractères (table de codage)

Premier codage (Hollerith)

1890 : stockage du recensemement des USA sur des cartes de Jacquard

1924 : fondation d'IBM, monopole de la production des cartes perforées

1950 : Cartes perforées : 80 colonnes, 12 lignes

http://www.maxmon.com/punch1.htm

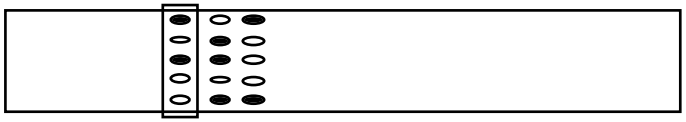
Un caractère par colonne, 3 trous pour définir un caractère.

Code : Telex

Système électromécanique, stockage sur bande papier perforée.

5 trous sur la largeur de la bande => 5 bits par caractère => 32 caractères

=> 26 lettres majuscules + ponctuation



Utilisation en informatique : mémoire externe

(10 car. --> env. 5cm de bande ; 1Mc --> 5km)

Codages

6 bits

Permet de représenter les 26 lettres majuscules latines, les chiffres 0..9, les symboles de ponctuation: , : ; . ( ) etc.

Suffisant pour écrire des programmes et imprimer des résultats.

7 bits

128 caractères

Standard ASCII (American standard for communication and information interchange)

adapté à la langue américaine :

- lettres majuscules et minuscules sans accents,
- chiffres, symboles de ponctuation.

Codages suite

8 bits

256 caractères possibles

3 "standards"

IBM-PC : majuscules, minuscules, lettres accentuées occidentales

Apple Macintosh : idem mais autre codage

ISO-xxx

Ensemble de standards

adaptés à différents groupes linguistiques

(p.ex. ISO-Latin-1 => langues latines)

Utilisé sur les Unix modernes, et sous Windows

Standard Unicode

- Capacité d'encoder tous les caractères du monde
- Codage sur 16 bits --> 65536 caractères
- Mécanisme d'extension à 1 mio car.
  - caractères historiques (hiéroglyphes, etc.)
  - notations communes (math, physique, etc.)
- Compatible avec ISO/IEC 10646.

Caractères traités

- Latin, Greek, Cyrillic, Armenian, Hebrew, Arabic, Devanagari, Bengali, Gurmukhi, Gujarati, Oriya, Tamil, Telugu, Kannada, Malayalam, Thai, Lao, Georgian, Tibetan, Japanese Kana, modern Korean Hangul, Chinese/Japanese/Korean (CJK) ideographs.
- Bientôt: Ethiopic, Canadian Syllabics, Cherokee, additional rare ideographs, Sinhala, Syriac, Burmese, Khmer, and Braille.
- Ponctuation, diacritiques, math., technique, flèches, dingbats
- Signes diacritiques de modification de prononciation (n + '~' = ñ)
- 18,000 codes en réserve

Elements de texte et de codage

Element de texte

≠ caractère

En espagnol historique "ll" compte comme un seul élément => influence sur le tri des mots.

>>> Dépend du traitement, ne peut être défini dans l'absolu

Element de codage (caractères)

Unicode définit des éléments de code indépendants du traitement

Correspondent aux éléments de texte les plus fréquents

"ll" --> deux codes: 'l' + 'l'

chaque lettre maj. et min. est un élément de code

L'application est responsable de la (re)formation des éléments de texte.

Ecriture hexadécimale des codes

On écrit la valeur numérique des codes en hexadécimal (base 16)

notation : **U+xxxx**

L'hexadécima se convertit facilement en binaire

1 chiffre hexadécimal = 4 bits

```
0 --> 0000, 1 --> 0001, 2 --> 0010, 3 --> 0011,
4 --> 0100, 5 --> 0101, 6 --> 0110, 7 --> 0111,
8 --> 1000, 9 --> 1001, A --> 1010, B --> 1011,
C --> 1100, D --> 1101, E --> 1110, F --> 1111
```

Exemple U+**B2F5** --> **1011**0010**1111**0101

Caractères et Glyphs

Différence fondamentale :

valeur d'un code ≠ son affichage sur l'écran

Notion de caractère abstrait

"LATIN CHARACTER CAPITAL A"

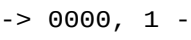
"BENGALI DIGIT 5"

Glyph = la marque faite sur un écran ou sur papier

représentation visuelle d'un caractère

Unicode ne definit pas les glyphs

ne spécifie pas la taille, forme, orientation des caractères sur l'écran

a  ==> un seul caractère Unicode (U+0065)

Caractères composites (â)

Formé de

- une lettre de base "a" -- qui occupe un espace
- une marque "^" (ou plusieurs) -- sur le même espace

Unicode spécifie

- l'ordre des car. pour créer un composite
- la résolution des ambigüités
- la décomposition des caractères précomposés

  "ü" peut être encodé par U+00FC (un seul car. 16-bits)

  ou bien décomposé en U+0075 + U+0308 ("u"+"¨").

  => compatibilité avec ISO-Latin-1 etc.

Intérêt de la décomposition : travailler sur les caractères de base uniqement (p.ex. pour le tri, ou la recherche de mots)

Définition des codes

Inclusion de standards précédents (0 .. 256 = Latin-1).

Notion de script :

- système cohérent de caractères (related characters)
- regroupe plusieurs langues
- évite la duplification (p.ex. chinois, japonais, koréen => script CJK)

Texte = séquence de codes correspondant à l'ordre de frappe au clavier

Caractères de changement de direction

Assignation des codes

Un nombre de 16 bits est assigné à chaque élément de code du standard.

Ces nombres sont appelés les valeurs de code

  U+0041 = nombre hexadécimal 0041 = décimal 65 représente le caractère "A" .

Chaque caractère reçoit un nom

  U+0041 s'appelle "LATIN CAPITAL LETTER A."

  U+0A1B s'appelle "GURMUKHI LETTER CHA."

  (standard ISO/IEC 10646)

De blocs de codes de taille variable sont aloués aux scripts

L'ordre "alphabétique" est si possible maintenu

Base de donnée de codes

0 Code value
1 Character Name.
2 General Category
3 Canonical Combining Classes
4 Bidirectional Category.
5 Character Decomposition
6 Decimal digit value
7 Digit value.
8 Numeric value
9 ? "mirrored"
10 Unicode 1.0 Name
11 10646 Comment field.
12 Upper case equivalent mapping.
13 Lower case equivalent mapping
14 Title case equivalent mapping

Quelques caractères

```
0041;LATIN CAPITAL LETTER A
     ;Lu;0;L;;;;;N;;;;0061;
005E;CIRCUMFLEX ACCENT;Sk;0;ON;<compat> 0020
     0302;;;;N;SPACING CIRCUMFLEX;;;;
0F19;TIBETAN ASTROLOGICAL SIGN SDONG TSHUGS;
     Mn;220;ON;;;;;N;;dong tsu;;;
112C;HANGUL CHOSEONG KAPYEOUNSSANGPIEUP;
     Lo;0;L;<compat> 1107 1107 110B;;;;N;;;;;
1EE4;LATIN CAPITAL LETTER U WITH DOT BELOW;
     Lu;0;L;0055 0323;;;;N;;;;1EE5;
FC64;ARABIC LIGATURE YEH WITH HAMZA ABOVE WITH
     REH FINAL FORM;Lo;0; R;<final> 0626 0631;
     ;;;N;;;;;
```

ftp://ftp.unicode.org/Public/2.1-Update3/UnicodeData-2.1.8.txt

Formes d'encodage

Représenter les codes Unicode de manière plus ou moins compacte.

En fonction des besoins

- en espace
- en temps

Conversions sans perte entre les différents codages.

UTF-8

Codage à longueur variable

U+0000 .. U+007F	==> 1 byte (même codes que ASCII)
U+0080 .. U+07FF	==> 2 bytes
U+0800 .. U+FFFF	==> 3 bytes
U+10000 .. 0x1FFFFF	==> 4 bytes
U+200000 .. U+03FFFFFF	==> 5 bytes
U+4000000 .. U+7FFFFFFF	==> 6 bytes

Avantage : pas (peu) de réécriture des programmes qui traitent le code ASCII

Taille variable => **indicateur de longueur** (la machine ne peut pas deviner)

<longueur> <valeur> <longueur> <valeur> etc.

Méthode de Codage UTF-8

Le nombre de 1 au début du premier byte donne la taille du code en bytes

Les bytes 2 à n commencent tous par **10**.

U-00000000 - U-0000007F: **0xxxxxxx**

U-00000080 - U-000007FF: **110xxxxx 10xxxxxx**

U-00000800 - U-0000FFFF: **1110xxxx 10xxxxxx 10xxxxxx**

U-00010000 - U-001FFFFF: **11110xxx 10xxxxxx 10xxxxxx 10xxxxxx**

U-00200000 - U-03FFFFFF: **111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx**

U-04000000 - U-7FFFFFFF:

1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

Ref: http://www.cl.cam.ac.uk/~mgk25/unicode.html#utf-8

Exemple

a é 本 b ñ 語 に

61 c3 a9 e6 9c ac 62 c3 b1 e8 aa 9e e3 81 ab

011000001 **11**000011 **10**101001 **111**00110 **10**011100 **10**101100 **0**1100010

11000011 **10**110001 **111**01000 **10**101010 **10**011110 **111**00011 **10**000001 **10**101011

Pourquoi **10** ?

10 ne peut être le début d'un caractère.

Permet de se synchroniser sans revenir au début du texte.

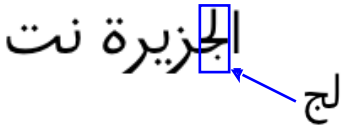
Exemple

ت d8 aa

شنت d8 b4 d9 86 d8 aa

! sens d'écriture ≠ ordre dans le fichier Unicode

mêmes codes mais changement de forme des caractères



UTF-16

- caractères 16-bits
- paires de 2 x 16-bits pour extension

<< UTF-16 is popular in many environments that need to balance efficient access to characters with economical use of storage. It is reasonably compact and all the heavily used characters fit into a single 16-bit code unit, while all other characters are accessible via pairs of 16-bit code units. >>

UTF-32

<< UTF-32 is popular where memory space is no concern, but fixed width, single code unit access to characters is desired. Each Unicode character is encoded in a single 32-bit code unit when using UTF-32. >>

Dans les langages de programmation

C :

type char = 8 bits

pas de codage spécifié ==> dépend de la machine utilisée

Java :

type char = 16 bits Unicode

Transcodage de/vers UTF-8, UTF-16 et autres lors de la lecture/écriture des fichiers

- classes Reader et Writer