情報処理 II 第 5 回

情報の電子化 (1) 文字コードとテキスト・ファイル

桂田 祐史 (かつらだ まさし)

2005 年 5 月 26 日

ホームページは http://www.math.meiji.ac.jp/~mk/syori2-2005/

これから数回、情報をどのようにコンピューター上で表現するかということをテーマにする。主に UNIX (Linux) 上の小さなプログラムを用いた実験を通して説明する。授業開始時に Linux 環境にログインしておこう[1]。やり方は以前説明しておいた:

> 『2005 年度新システム』
> http://www.math.meiji.ac.jp/~mk/syori2-2005/jouhousyori2-2005-00/node2.html

今回は実験しながら、テキスト・ファイルの構造、特に文字コードについて学ぶ。
(今回の内容は「例年通り」であるが、新しいシステム (Linux) に不慣れな人も多いと考えられるので、実際にコンピューターを操作して試す部分はなるべく時間を取って行なう。あるいは、予定してある内容を消化しきれないかもしれないことを最初に断っておく。)

1 連絡事項

- 課題 2 のレポートの講評 (説明) は次回にする (昨日が締め切りではあるけれど、今週いっぱいまで待ちます)。
- 先週インターネット講習会を受講した人は、もう利用資格が得られているはず。チェックは次のページでできる。

  > 『受講者チェックシステム』
  > http://www.meiji.ac.jp/mind/tool/internet-license/

  学外の WWW ページを見るための方法は以下の通り。

  (1) Windows 環境では、ログオンする毎に、[スタート] メニューの [レベル 2 利用資格認証] を選択・実行し、ユーザー名とパスワードを入力する。

[1] 今日の内容は Linux でなくても実行することはそれほど難しくはないが、次回の内容は Linux でないと難しいので、慣れるため今日から Linux を使うことにする。

(2) Linux 環境の Mozilla では、一度だけ、以下に説明する「プロキシーの設定」を行なう。[編集] → [設定] → [詳細] → [プロキシ] で「自動的にプロキシを設定する」を選択し、URL として `http://www.isc.meiji.ac.jp/proxy.pac` を指定する。

2 今週来週のための設定

以下に登場する qkc, nkc を使ってみたければ、`~/.cshrc` に (emacs[2]などで)

```
set path=($path /usr/meiji/pub/Linux/bin)
```

という行を書き加えてからターミナルを起動すること。

次回はキーボード入力が多いので、tcsh または bash を使うように設定しておくことを強く勧める。

```
a308-00% chsh
ee48099 のシェルを変更します。
Password:
新しいシェル [/bin/csh]: /bin/tcsh
a308-00%
```

3 コンピューターのファイル — 万物はビットである

3.1 デジタル、ビット、バイト

現在の一般的なコンピューター[3]の処理するデータは**デジタル**[4]である。

より具体的に言うと、

コンピューター上のすべてのデータは、ビット[5] (bit) の有限列である。

実際には、複数のビットを適当な大きさにまとめて処理することが多い。例えば

- 表示する際には、3 桁, 4 桁ごとに区切って 8 進数, 16 進数とする[6]。
- データの格納や移動の際には、バイト[7] やワード[8]という単位が使われることが多い。

---

[2]emacs の使い方は知っているはずだが、`emacs .cshrc &` とする。

[3]コンピューターの古い呼び方は — electric digital computer (電子計数型計算機) — という。アナログ計算機というものもあった。

[4]デジタル (digital) — もともとは「指の」という意味だが、ここでは「(離散的な) 数字の」という意味。

[5]0 または 1 の二つの状態のいずれであるかを示す、情報量の単位をビットと言う。電子回路との相性がいい。

[6]コンピューターの世界では、16 進法の数字として '0' 〜'9', 'A', 'B', 'C', 'D', 'E', 'F' を用いるのが普通。

[7]バイト (byte) — 通常は 8 ビットのこと。 2^8 通りの状態を区別出来るので、例えば 0 〜255 までの整数値を割り当てて読むことがよく行なわれる。ときどき「コンピューターは 2 進法で計算すると言うけれど、256 進法ではないか？」と思うこともある。16 進法では 2 桁で表示することになる ($\because 2^8 = 16^2$)。

[8]ワード (word) — そのコンピューターにとって都合のよいデータの大きさで、普通はバイトの整数倍の大きさ。1 ワードが 16 bits, 32 bits, 64bits などの場合が多い。

コンピューター上の情報はすべてファイルとして保存できるが、もちろん中身は数列である。特に Windows や、UNIX では、

ファイルはバイトの有限列である (ファイルは数列である)

と言って構わない。

色々なデータ (文書、数値データ、静止画、動画、音声, etc.) を記録したファイルがあるが[9]、バイトの列であることに変わりはない。ファイルの保存、コピー、通信による転送など、共通の操作で扱うことが出来る。

3.2 ファイルの大きさを調べる

UNIX 環境では、ファイルの大きさ (サイズ) は、`ls -l` または `wc ファイル名` とすれば分かる (前者はバイト数、後者は文字数、単語数、行数を調べる)。

K, M, G, T, P, ..

国際単位系 (Le Systè me international d'Unité s, SI) で、大きな量を表すために、deca (da), hecto (h), kilo (k), mega (M), giga (G), tera (T), peta (P), exa (E), zeta (Z), yotta (Y) などの接頭語を補助的に用いる。コンピューターの世界でも、K= $1024 = 2^{10} \fallingdotseq 10^3$, M(メガ)= K^2 = 約 100 万, G(ギガ)= K^3, T(テラ)= K^4, P(ペタ)= K^5 などを用いる。身の回りにある記憶装置、記憶媒体 (メイン・メモリー、フロッピー・ディスク、ハードディスク、CD-ROM, MO, DVD, USB メモリ, 各種メモリカードなど) の容量の表示にはよく登場する。

3.3 レポート課題 X (予告)

以下のことを調べよ (〆切は 6 月末頃にする予定。X はそのとき適当な番号をつける。)。(ファイルのサイズについての感覚を身につけるのが目的である。)

(1) 情報科学センターの Windows 環境、Linux (UNIX) 環境上のファイルのサイズについて調べよ。バイト数以外に、自分の身近[10]にある記録媒体 (CD-R や USB メモリ, SD メモリカードなど、適当に選ぶ) にどれくらい入るかを記せ。

(a) 文書ファイル。
レポート、メイル、C プログラム、TEX のソース (`.tex`) など。
(印刷して何ページくらいの文書が何バイトになるか。)

(b) 実行可能プログラム
例えば C プログラムはコンパイル前と後でどうサイズが変わるか。
自分が普段使っているプログラム[11]のファイルのサイズは？
(emacs は主に C 言語で書かれているが、何行くらいになるか想像してみると面白い。)

[9] 余談だが、ファイルの種類をある程度まで自動的に判別するコマンド `file` がある。`file *` としてみよう。
[10] 以前は「フロッピー・ディスクに」としたが、もう身近なものではないのだろう。
[11] コマンドの実体がどこにあるか (パス名) は、UNIX では、which コマンドを使って調べられる。例えば、`which emacs` とすると、emacs コマンドのパス名は `/usr/bin/emacs` であることが分かる。

(c) (もし出来れば) 画像ファイル、音声ファイルなど。
(図の大きさ、色数も分かれば調べる。この問に関しては、情報科学センターのマシンでなく、自分の持っているデジカメや携帯電話で調べる方が面白いかも知れない。)

(d) 現在、自分が持っているファイルの総量。それは自分のホームディレクトリィのあるディスクの全容量の何 % に相当するか？

```
du -ks ~              ← ホームディレクトリィ下のファイルの総量
du -ks ~/.snapshot    ← .snapshot の下のファイルの総量
df -k                 ← そのマシンにマウントされているディスクの状態
```
(それぞれで何が分かるかは授業中に話す。聞き漏らした場合は自力で！)

あるいは Windows での使用がメインの場合、マイドキュメント・フォルダの下にあるファイルの総量を調べるのが良いかも知れない。

(2) 自分が持っている本を一冊選び、その文字情報を記憶するファイルを作った場合、サイズはどれくらいになるか計算せよ。フロッピー・ディスクに記憶する場合、何枚必要か？また CD-ROM (容量 650MB 程度) には何冊分記憶できるか。

注意: 結果は K, M などを適切に選んで表現すること。例年間違えて結果に 1000 倍の差が出る人が少なからずいる ($1000^2 = 100$ 万 倍の間違いもあった…)。

4 テキスト・ファイル

4.1 最初に大事な言葉の意味

テキスト・ファイルとは

コンピューター上のファイルのうち、普通の意味で「読める」字で出来ているファイルのことを**テキスト・ファイル** (text file) という[12]。

文字コードとは

コンピューターでは、文字も数で表す。このためには「この数はこの文字」とルールを決める必要がある。文字を表現するため、表現したい文字の集合と、数の集合を選び、両者の間の 1 対 1 の写像を定める。このとき、ある文字に対応する数を、その文字の**文字コード**と呼ぶ。

4.2 英数字の文字コード (ASCII)

まず emacs を使って、

[12]Windows の世界では、拡張子 "`.txt`" をつけるファイルのことであるが、これは OS にとらわれない重要な概念であるので、時間をかけて解説する。

ascii.txt

```
012
ABC abc
```

という 2 行からなるファイルを作成し、以下のコマンド[13]を実行してみよう。

```
 isc-xas06% od -tx1c ascii.txt

0000000 30 31 32 0a 41 42 43 20 61 62 63 0a
          0   1   2  \n   A   B   C       a   b   c  \n
0000014
```

(od -tx1　ファイル名 とすると、ファイルの各バイトの文字コードが 16 進数で表示される。)
次の事が分かる[14]。

- 数字 '0', '1', ··· の文字コードは 0x30, 0x31, ···
- 英字 'A', 'B', ··· の文字コードは 0x41, 0x42, ···
- 英字 'a', 'b', ··· の文字コードは 0x61, 0x62, ···
- 行の終りは、'\n' という一つの文字 (文字コードは 0x0a) で表される。
- 空白 ' ' も一つの文字 (文字コードは 0x20) で表される。

(相当に不正確な言い方だが) このような「半角」英数字の文字コードはASCII (アスキー) というアメリカの規格で定められ、それが色々な規格に「輸入」されている。

[13] これまでは、od -cx ファイル名 とせよ、としていたのだが、これだと環境により結果が異なるので、od -tx1c ファイル名 とすることにした (古い od で使える？)。
[14] ここでは 16 進数を、先頭に "0x" をつけることで表した (C 言語の世界の習慣)。

4.3 ASCII コード表を作ろう

print_code_table.c

```
/*
 * print_code_table.c --- ASCII コード表 (印字可能な文字のみ) を表示する。
 *   コンパイル: gcc -o print_code_table print_code_table.c
 *   実行:       ./print_code_table
 *   あるいは: gcc print_code_table.c ; ./a.out
 */

#include <stdio.h>

int main()
{
    int code;
    /* 0x20 (==32) から 0x7e (==126) までの数をコードとする文字を扱う */
    for (code = 0x20; code <= 0x7e; code++) {
        /* 4 つおきに改行 */
        if (code % 4 == 0)
            printf("\n");
        /* 16 進数, 10 進数, 文字 として表示 */
        printf("0x%02x (%3d): %c    ", code, code, code);
    }
    printf("\n");
    return 0;
}
```

というプログラム (WWW ページに載せてあるので、WWW ブラウザー (例えば Mozilla) で例えばホームディレクトリィ) にセーブするとよい) をコンパイル&実行してみよう。

C 言語の豆知識

- a % b で a を b で割った余りを表わす。
- putchar(整数式); あるいは printf("%c", 整数式); でその整数を文字コードとする文字が表示される。
- printf() で 16 進数を表示するには %x という書式を用いる (%d は 10 進数)。

4.4 日本語の文字コード

日本語の文字コードは JIS (日本工業規格) で決められている。日本語における文字数が多いため[15]、1 文字を表すのに 16 ビットを用いる。

例えば「桂」という文字の JIS コードは 0x374b である。

しかし、ファイルの中に 0x37, 0x4b というバイト列をそのまま入れたのでは、ASCII の '7', 'K' と区別がつかない。両者を混在させるには何らかの工夫が必要になる。

情報科学センターの Solaris (UNIX) 環境では**日本語 EUC** という文字コードを用いている。やはり emacs を用いて

[15]ちなみに JIS の情報交換用漢字符号系コードの第一、第二水準の漢字は約 7000 個弱ある。JIS にはいくつかのバージョンがある。また第一、第二水準以外のものもあるが、あまり普及していない。

kanji.txt

桂田 祐史

というファイルを作成して、od -x1 kanji.txt としてみると、「桂」が 0xb7, 0xcb という 2 バイトで表現されていることが分かる。これは JIS コードの上下 8 ビットにそれぞれ 0x80 を加えたものになっている:

$$\texttt{0x37} + \texttt{0x80} = \texttt{0xb7}, \quad \texttt{0x4b} + \texttt{0x80} = \texttt{0xcb}$$

(もともと ASCII で用いている数値は 0x7f までで、0x80 以上の数値は使われずに空(あ)いている。その部分に JIS コード表の文字を埋め込んだことになっている。)

日本語 EUC 以外にも、ASCII の文字と日本語の文字を混在を可能にした文字コードはいくつかある。

準備: nkf を用いて kanji.txt の文字コードを変換する

```
a308-00% nkf -j kanji.txt > kanji-jis.txt
a308-00% nkf -s kanji.txt > kanji-ms.txt
```

(nkf については、付録 A を見よ。)

通称「JIS 漢字」 通信により情報を交換するために作られた国際的な規格に従ったもの。文字コードを切り替えるために、目印となる特別な文字列[16]を用いる。電子メイルなどで、日本語メッセージの通信をするときに最もよく使われる ISO-2022-JP (後述) の基礎となった。実は kterm では JIS 漢字のデータも普通に表示できる。

```
a308-00% cat kanji-jis.txt
桂田 祐史
a308-001% od -tx1 kanji-jis.txt
0000000 1b 24 42 37 4b 45 44 1b 28 42 20 1b 24 42 4d 34
0000020 3b 4b 1b 28 42 0a
0000026
a308-00%
```

「桂」の JIS コードである 0x37, 0x4b が現われている。

通称「MS 漢字 (シフトジス)」 パソコンの世界のために作られた規格 (Windows, Mac などで採用されている)。「半角カタカナ」も効率的に埋め込まれている。情報科学センターのワークステーションの通常の設定では直接表示できない (ただし emacs では扱える)。

[16]漢字に切り替えるために 0x1b, 0x24, 0x42 を、ASCII に戻すために 0x1b, 0x28, 0x42 を用いる。

```
a308-00% cat kanji-ms.txt
jc S （いわゆる文字化け）
a308-00% od -tx1 kanji-ms.txt
0000000 8c 6a 93 63 20 97 53 8e 6a 0a
0000012
a308-00%
```

「桂」が 0x8c, 0x6a となっているが、どういうルールで変換されているかはちょっと分かりづらい。付録 D に JIS を MS 漢字に直す C の関数をあげておく。

訂正

2005 年 6 月 2 日現在、情報処理教室の設定では、emacs で普通に日本語テキスト・ファイルを作成すると、JIS 漢字コードのファイルができる (これはちょっと問題だと思うのだが…)。そこで以下の手順で作業すること。

```
a308-01% emacs kanji-jis.txt &
a308-01% nkf -e kanji-jis.txt > kanji.txt
a308-01% nkf -s kanji-jis.txt > kanji-ms.txt
a308-01% od -tx1 kanji.txt
a308-01% od -tx1 kanji-jis.txt
a308-01% od -tx1 kanji-sjis.txt
```

腕試し用プログラミング課題 1

print_code_table.c が ASCII コード表を表示するように、漢字のコード表を表示するプログラム print_kanji_table.c を作れ。レポートを送るときは Subject (件名) は “腕試し 1” として下さい。第 E 節のプログラムを参考にするとよい。

4.5 文字コードをいじってみよう

WWW ページに mycat.c というプログラムを載せてある。これは cat コマンドの真似をして、ファイルの内容を標準出力 (通常は画面) に出力するだけのプログラムである。

mycat.c をコンパイルする

```
a308-00% gcc -o mycat mycat.c
a308-00% ./mycat mycat.c
```

このプログラム mycat.c を読んでみよう。中の print_file() という関数を書き換えると、色々なことができる。例えば

日本語 EUC に対応した mydump.c の print_file

```
void print_file(FILE * fp)
{
    int c, c2;
    /* ファイルの終りまで一文字ずつ c に読み込み、標準出力に書き出す */
    while ((c = getc(fp)) != EOF) {
        if (c >= 0x80) {
            /* 0x80 以上だったら漢字の 1 バイト目だと判断して、
               もう 1 バイト読んで、まとめて出力する。 */
            c2 = getc(fp);
            printf("0x%02x 0x%02x: %c%c\n", c, c2, c, c2);
        } else if (c < 0x20 || c == 0x7f)
            /* 0x20 未満または 0x7f の場合は文字コードのみ表示 */
            printf("0x%02x\n", c);
        else
            /* それ以外の場合は文字コードと、その文字自身を出力 */
            printf("0x%02x: %c\n", c, c);
    }
}
```

```
a308-00% cat ascii_and_kanji.txt
I am 桂田祐史.
a308-00% ./mydump ascii_and_kanji.txt
0x49: I
0x20:
0x61: a
0x6d: m
0x20:
0xb7 0xcb: 桂
0xc5 0xc4: 田
0xcd 0xb4: 祐
0xbb 0xcb: 史
0x2e: .
0x0a
a308-00%
```

腕試し用プログラミング課題 2

ファイルの中の英小文字を英大文字に変換するプログラムを作りなさい。レポートを送るときは Subject (件名) は “腕試し 2” として下さい。(ヒント: 小文字、大文字のコードはそれぞれ連続していて、アルファベット順に並んでいる。)

腕試し用プログラミング課題 3

ファイルのバイト数、行数を数えるプログラムを作りなさい。(‘\n’ (文字コード 0x0a) の個数を行数と考えることにする。) レポートを送るときは Subject (件名) は “腕試し 3” として

下さい。

5　レポート課題 3

自分の名前を構成する各々の文字 (桂田祐史なら「桂」,「田」,「祐」,「史」の 4 文字) の JIS コード, EUC コード, SJIS コードを調べよ。宛先は syori2@math.meiji.ac.jp で、件名は「情報処理 II レポート課題 3」。締め切りは 6 月 15 日 (水曜) にする予定。

(以下は付録)

A　日本語の文字コードの変換

UNIX 上のコマンドには自動的に文字コードを判別して必要な処理をしてくれるものがあるが (emacs, less 等)、時にはユーザーが意識的に変換することが必要になる。

それほど難しい作業でもないのでフリーソフトがある。二つほど紹介する。

nkf (Network Kanji code conversion Filter) UNIX では定番。

`nkf -e` ファイル名 で日本語 EUC に変換したものを標準出力に書き出す。

`nkf -j` ファイル名 で JIS 漢字コードに変換したものを標準出力に書き出す。

`nkf -s` ファイル名 で MS 漢字コードに変換したものを標準出力に書き出す。

電子メールで使われる MIME のデコードもできる。

`nkf -v` でオプションの一覧が表示される。

`kanji.txt` を JIS 漢字に変換した `kanji-jis.txt` を作る

```
a308-00% nkf -j kanji.txt > kanji-jis.txt
```

qkc (Quick KANJI code Converter) Windows 版もある。行末の変換もしてくれる。

`qkc -eu` ファイル名 で日本語 EUC, 行末を UNIX 形式に変換する。

`qkc -ms` ファイル名 で MS 漢字, 行末を MS-DOS (Windows?) 形式に変換する。

`kanji.txt` を MS 漢字に変換した `kanji-ms.txt` を作る

```
a308-00% cp kanji.txt kanji-ms.txt
a308-00% qkc -ms kanji-ms.txt
```

B　インターネットで使って良い文字悪い文字 (日本語)

コンピューター上で色々な日本語文字が使えるようになっていて、テキスト・ファイルを作ることが出来るが、インターネット (電子メイル、WWW ページ、ネットニュースなど) で用いる場合、すべての文字を使えるわけではない。

- 電子メイル、ネットニュースでは、**ISO-2022-JP** と呼ばれる文字コードが使える。
  インターネットの作法である RFC で定義されている。
  → `http://www.noge.com/koba/network/RFC/rfc1468.html`
  元々は日本のインターネットのルーツである JUNET で利用されて来たもの。
  ASCII, JIS X 0201-1976, JIS X 0208-1978, JIS X 0208-1983 という 4 種類の文字コードをエスケープ・シーケンスというバイト列でスイッチする。
  - ASCII はアメリカの規格 (キーボードから直接打ち込める英数字・記号)。
  - JIS X 0201 はその日本版 (円記号とバックスラッシュなど ASCII と異なる)。
  - JIS X 0208 はいわゆる JIS 第一、第二水準の漢字、ひらがな、カタカナ、その他記号。
- 「外字」は使えない (論外)。
  (外字については後述。)
- 機種依存文字も要注意。後々まで残る可能性のあるデータの表現には使うべきではない。
  (コンピューターやソフトウェアのメーカーが作った文字。普通の人には、JIS で正式に定義された文字と見分けがつきにくいかも知れない。丸つき数字、ローマ数字、携帯電話の絵文字などが有名。)
- JIS X201 の右半面 (俗称「半角カナ[17]」) は RFC に違反しているので、メイルやネットニュースでは使用してはいけない。

C 参考 URL

1. 『日本語フォントや文字コードについて』

   `http://www.ryukyu.ad.jp/~shin/jdoc/`

   なかなか充実したリンク集。このページを見るだけで、文字コードの問題がなかなか複雑であることを感じ取れる :-)

2. 『ほら貝』by 加藤弘一

   `http://www.horagai.com/`

   日本語の文字コードの話を調べて、本 (『電脳社会の日本語』, 文藝春秋, 2003, `http://www.horagai.com/www/salon/works/denno.htm`) も出している人で、今は「文字コードからは足を洗った」そうですが、どっこい WWW ページ上で『文字コードから見た住基ネットの問題点』など書いて発表していらっしゃいます。

[17] 昔の MS-DOS パソコンやワープロ専用機では、これらの文字が他の日本語文字の半分の幅で表示されたため、こう呼ばれた (←→全角)。元々は印刷業界の言葉であったとか。しかし、そもそも文字コードの JIS 規格には「半角」という言葉はない。文字がどう表示されるかは、利用する環境によるので規格の範囲外である。

D　JIS を MS 漢字に直す関数

jis2sjis.c

```
/* JIS コード c1, c2 を MS 漢字コード s1, s2 に変換する */
void jis2sjis(int c1, int c2, int *s1, int *s2)
{
    if (c1 & 1) {
        c1 = (c1 >> 1) + 0x71;
        c2 += 0x1f;
        if (c2 >= 0x7f)
            c2++;
    }
    else {
        c1 = (c1 >> 1) + 0x70;
        c2 += 0x7e;
    }
    if (c1 > 0x9f)
        c1 += 0x40;
    *s1 = c1;
    *s2 = c2;
}
```

E　「桂」を文字コードを用いて表示する

print-katsura.c

```
/*
 * print-katsura.c --- 「桂」の文字を JIS コードを元にして EUC で表示
 */

#include <stdio.h>

int main()
{
  int j1, j2, e1, e2;
  j1 = 0x37; j2 = 0x4b;
  e1 = j1 + 0x80; e2 = j2 + 0x80;
  printf("%c%c\n", e1, e2);
  return 0;
}
```

F　文字コードおもちゃ箱

主に清水 [1] を参考にした。

F.1　ASCII — 最初の、かつ最も重要な情報交換用文字コード

コンピューター創成期は、メーカー各社が独自の文字コードを用いていた。

ANSI (American National Standards Institute[18]) が情報交換用の文字コードとして **ASCII** (7-Bit Amarican National Standard Code for Information Interchange, オリジナルは 1963) を制定した。

F.2 ISO 646 — ASCII のローカライズ

ヨーロッパ各国で、それぞれの事情に応じて、ASCII を小修正した文字コードができ、それが **ISO** (International Organization for Standardization[19]) でも **ISO 646** (ISO/IEC 646:1991 Information technology — ISO 7-bit coded character set for information interchange) として規格化された (1973)。

具体的には 0x23 (#), 0x24 ($), 0x40 (@), 0x5b ([), 0x5c (\), 0x5d (]), 0x5e (^), 0x60 (‘), 0x7b ({), 0x7c (|), 0x7d (}), 0x7e (~) の 12 箇所のコードが入れ替え可能になっている。

F.3 ISO 2022

(準備中)

F.4 ISO 8859 — ASCII の 8 ビット拡張 (欧米人ならこれでオッケー)

8 ビットの前半 (MSB が立っていない範囲) では ASCII のままで、後半に各国語固有の文字を収録した ISO 8859 が規格化された。特に最初に制定された **ISO 8859-1** (1987) はフランス語やドイツ語などの西ヨーロッパの文字をまとめて収録してあり、

F.5 JIS C 6220, JIS X 0201 — カタカナを使おう

1. 7 ビットのままで、ASCII のアルファベットとカタカナを切り替て用いる。SI (Shift In, 0x0e), SO (Shift Out, 0x0f) という文字で切り替える。SO でカタカナ、SI でローマ字となる。

2. 8 ビットで ISO 8859 流に、後半部分 (MSB が立っている部分) にカタカタを収録した。

F.6 いわゆる JIS 漢字 (JIS C 6226, JIS X 0208) — 漢字を使おう

1978 年、平仮名と漢字を扱うために JIS C 6226 が制定された。0x21〜0x7e までの 94 個を二つ組み合せた $94^2 = 8836$ 個の文字を定義できる領域を確保してある。

1 バイトの文字コード (例えば JIS C 6220) と、2 バイトの文字コード (例えば JIS C 6226) を混在させるために、**エスケープ・シーケンス**と呼ばれる文字列で切り替えを行なう。

- JIS C 6226-1978

[18] http://www.ansi.org/
[19] http://www.iso.ch/

- JIS X 0208-1983
- JIS X 0208-1990
- JIS X 0208-1997

F.7 JIS X 0212-1990 — 補助漢字

F.8 JIS X 0213-2000 — 第三、第四水準漢字

F.9 多漢字の試み: TRON コード, 今昔文字鏡, GT 書体

F.9.1 TRON コード

東京大学の坂村健が中心となった **TRON プロジェクト**(`http://www.tron.org/`) で規定している文字コードで、150 万字収納可能な領域を用意し、世界中で利用されているすべての文字を収録することを目標に作られたもの。

例えば「超漢字」という BTRON 準拠の OS で使うことができる。超漢字 3 (2001) では約 17 万字。

F.9.2 今昔文字鏡

今昔文字鏡[20]

F.9.3 GT 書体

F.10 ISO 10646

F.11 Unicode

F.12 ISO 10646-1:1993 = Unicode 1.1

参考文献

[1] 清水哲郎, 図解でわかる文字コードのすべて, 日本実業出版社 (2001).

[2] 加藤弘一, 電脳社会の日本語, 文春新書 094 (2000).

[20] `http://www.mojikyo.org/`