ななちゃんのIT教室

クリじい探検隊 テキストファイル探検の巻

by nara.yasuhiro@gmail.com

ななちゃんが
文字化けの謎解きに挑戦するという お話

第 0.7 2017 年 5 月 16 日

フリー素材
http://freeillustration.net

いらすとやフリー素材
http://www.irasutoya.com/

もくじ

第1回 探検に出発
第2回 UTF-8 の海へ
第3回 UTF-16 の海へ
第4回 EUC と Shift-JIS の海へ
第5回 JIS（ISO-2022-JP）の海へ
第6回 Unicode とは
第7回 文字コードの歴史
第8回 文字化けトリビア(1) 「 ¥ 」と「 \ 」
第9回 文字化けトリビア(2) Shift_JIS コードの悲劇
おまけ： JavaScript（ブラウザ）を使って、文字コードを調べる

第1回　探検に出発

クリ：クリじいじゃ。よろしくな。

なな：コロンブスに似ている！？

クリ：アメリカ大陸を発見したクリストファー・コロンブスはワシの 20 代前の先祖での。ワシもクリストファーなのでクリじいじゃ。新大陸発見に臨んだ行動力と、インドと誤解したおっちょこちょいな点も似ておる。

なな：何を探検するの？

クリ：いろいろなファイルの中身を探検するのじゃ。

なな：船にのって出かけるの？

クリ：「16 進ダンプ」というスグレモノを使うのじゃ。 UNIX の「% od –x」とか、Windows のフリーソフトとかあるじゃろ。

なな：どこに出かけるの？

クリ：下のような内容をエディタで入力して、ファイルに書き込み、それを「16 進ダンプ」で調べるのじゃ。

```
１２１２１２
カガｶﾞ
```

「16 進ダンプ」で読み込むと、こんなのが見えてくる。

```
---------|-0--1--2--3--4--5--6--7--8--9--A--B--C--D--E--F-|
00000000|ef bb bf ef bc 91 ef bc 92 31 32 ef bc 91 ef bc |
00000010|92 0d 0a e3 82 ab e3 82 ac ef bd b6 ef be 9e 0d |
00000020|0a                                              |
---------|------------------------------------------------|
```

なな：なぜ「16進」なの？

クリ：1/0 との対応、つまりビットパターンが想像しやすいからじゃ。2 進数だと、桁数が長くなりすぎるので、16進がちょうといいんじゃ。1 バイトが 16 進数 2 桁に対応する。16 進数 6a は 2 進数で 0110 1010 とすぐに分かるが、10 進数の 131 の 2 進表現なんかすぐにはわからんからな。

なな：「16 進ダンプ」で、何が分かるの？

クリ：文字化けが発生する原因をつきとめるのじゃ。

なな：文字化けって、ウェブページの表示がおかしくなること？

```
｣ｱ｣ｲ 12｣ｱ｣ｲ
･ｫ･ｬ叱酌
```

みたいになってしまったり、「À」が 「ﾀ」になったり、「©」が 「ｩ」になったりするヤツね。

クリ：そうそう。

なな：なぜそうなるのかが分かるってこと。

クリ：まあな。とにかく、調べてみよう。

なな：後悔に出発ということね！

クリ：「後悔」じゃなくて、「航海」じゃろ。お前が文字化けしてどうするんじゃ。

なな：とにかく、出発！

第2回　UTF-8 の海へ

クリ: ななちゃん、まず、下のような内容をエディタで入力して、UTF-8 というコードでファイルに書き込んでから、それを「16 進ダンプ」で読み込んでみてくれんか。

```
１２12１２
カガｶﾞ
```

なな: は～～い。これでいいかしら。

```
--------|-0--1--2--3--4--5--6--7--8--9--A--B--C--D--E--F-|
00000000|ef bb bf ef bc 91 ef bc 92 31 32 ef bc 91 ef bc |
00000010|92 0d 0a e3 82 ab e3 82 ac ef bd b6 ef be 9e 0d |
00000020|0a                                              |
--------|------------------------------------------------|
```

クリ: 上出来じゃ。「ef」とか、「bb」とかが、ファイルの中身を 16進数で表現したものじゃ。 16進 2 桁で 8 ビットだから、全部で 33 バイトあるということじゃな。

なな: 暗号みたいね。これから、何が分かるの？

クリ: 「0d 0a」は、CR LF、つまり、「キャリッジリターン」（復帰） と 「ラインフィード」（改行） の印なんじゃ。ちなみに、UNIX では、LF だけ、Macintosh では CR だけ、Windows では両方を使うのが普通になっておる。昔のタイプライターの印字ヘッド（キャリッジ）という部品が、左端に戻って（リターン）、紙が 1 行（ライン）ぶん進む（feed=紙送り）ということ。ななちゃん、「0d 0a」 のところで、改行してみてくれんか。

なな: は～～い。

```
ef bb bf ef bc 91 ef bc 92 31 32 ef bc 91 ef bc 92
0d 0a
e3 82 ab e3 82 ac ef bd b6 ef be 9e
0d 0a
```

クリ: それから、1 行目の末尾と、はじめのほうに、全角の「１２」があるはずだから、それを分けるんじゃ。

なな: う～ん。

```
ef bb bf
ef bc 91       「１」
ef bc 92       「２」
31 32
ef bc 91       「１」
ef bc 92       「２」
0d 0a         CR LF
e3 82 ab e3 82 ac ef bd b6 ef be 9e
0d 0a         CR LF
```

くり: 前回の「１２」と「１２」の間に、半角の「12」があるはずじゃ。それから、残りを 3 バイトずつ束ねる。

なな：こうかしら。

```
ef bb bf
ef bc 91        「１」
ef bc 92        「２」
31              「1」
32              「2」
ef bc 91        「１」
ef bc 92        「２」
0d 0a           CR LF
e3 82 ab        「カ」
e3 82 ac        「ガ」
ef bd b6
ef be 9e
0d 0a           CR LF
```

クリ：そうじゃ。それから、半角の「ｶﾞ」は、実は、「ｶ」と、「ﾞ」に分かれるので、それぞれ割り当てる。

なな：ラジャー。

```
ef bb bf
ef bc 91        「１」
ef bc 92        「２」
31              「1」
32              「2」
ef bc 91        「１」
ef bc 92        「２」
0d 0a           CR LF
e3 82 ab        「カ」
e3 82 ac        「ガ」
ef bd b6        「ｶ」
ef be 9e        「ﾞ」
0d 0a           CR LF
```

クリ：そうそう。これで 解析完了。

なな：先頭の「ef bb bf」は？

クリ：このファイルが UTF-8 で書き込まれているという印なんじゃ。 半角の「1」や「2」は、 ASCII 文字コードというもので、下記のようになっておる。

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	**LF**	VT	FF	**CR**	SO	SI
10	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
20	SP	！	”	＃	＄	％	＆	’	（	）	＊	＋	，	－	．	／
30	0	**1**	**2**	3	4	5	6	7	8	9	：	；	＜	＝	＞	？
40	＠	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[	¥	]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{	｜	}	~	DEL

NUL とか SOH のようなのは「制御文字」という特殊コードで、LF とか CR もこの一種じゃ。そして、ASCII 文字以外は、1 文字を、3 バイトのデータで表しているということじゃな。

なな：よく分かんなけど、解析完了！

第3回　UTF-16 の海へ

クリ：ななちゃん、今度は、さっきと同じ内容をエディタで入力して、UTF-16 というコードでファイルに書き込んでから、それを「16 進ダンプ」で読み込んでみてくれんか。

```
１２12１２
カガｶﾞ
```

なな：は～～い。これでいいかしら。

```
--------|-0--1--2--3--4--5--6--7--8--9--A--B--C--D--E--F-|
00000000|ff fe 11 ff 12 ff 31 00 32 00 11 ff 12 ff 0d 00 |
00000010|0a 00 ab 30 ac 30 76 ff 9e ff 0d 00 0a 00       |
--------|------------------------------------------------|
```

クリ：そうそう。これを、さっきみたいに解析してみてくれんかの。

なな：は～～い。

```
ff fe
11 ff          「１」
12 ff          「２」
31 00          「1」
32 00          「2」
11 ff          「１」
12 ff          「２」
0d 00          CR
0a 00          LF
ab 30          「カ」
ac 30          「ガ」
76 ff          「ｶ」
9e ff          「ﾞ」
0d 00          CR
0a 00          LF
```

クリ：よくがんばった！

なな：こんどは、すべての文字が 2 バイトになっているわね。でも、CR が 0d 00 って、上下反対みたい。

クリ：こういうのを、「**リトルエンディアン**」というのじゃ。

なな：「リトルインディアン」？　「ワンリトル、ツーリトル、スリーリトルインディアンズ」？

クリ：本気で、そう誤解している人もいるが、「インディアン」じゃなくて「**エンディアン**」じゃ。それに、アメリカ大陸の原住民（ネイティブアメリカン）のことを、かつて、インディアンといっていたのは、ワシのご先祖のコロンブスがアメリカ大陸を発見した時に、インドであると勘違いしたからでの。原住民の方々は「我々はインド人じゃない！」と、不愉快に思うので、「インディアン」ということばは使わないように。

なな：わかりました。12 のことを、2 と 10 のように言うのがリトルエンディアン、10 と 2 のように言うのがビッグエンディアンということね。ビッグエンディアンのほうが自然だと思うけど。

クリ：インテル製の CPU が、リトルエンディアンのほうが高速処理できるというのと関係がある。

なな：それにしても、UTF-8 と UTF-16 とで、中身が全然違うのね！

第4回　EUC と Shift-JIS　の海へ

クリ：ななちゃん、また、同じ内容をエディタで入力して、今度は、EUC というコードと、Shift-JIS というコードでファイルに書き込んでから、それを「16 進ダンプ」で読み込んでみてくれんか。ついでに、解析も。

```
１２12１２
カガｶﾞ
```

なな：は～～い。これでいいかしら。まず、EUC。EUC は UNIX 向けということで、CR 無しです。CR+LF 付きの EUC も不可能ではないらしいけどね。全角文字は 2 バイトで、上位、下位バイトともに 80 以上なのが特徴のようね。

```
--------|-0--1--2--3--4--5--6--7--8--9--A--B--C--D--E--F-|
00000000|a3 b1 a3 b2 31 32 a3 b1 a3 b2 0a a5 ab a5 ac 8e |
00000010|b6 8e de 0a                                     |
--------|------------------------------------------------|
```

```
a3 b1   「１」
a3 b2   「２」
31      「1」
32      「2」
a3 b1   「１」
a3 b2   「２」
0a      LF
a5 ab   「カ」
a5 ac   「ガ」
8e b6   「ｶ」
8e de   「ﾞ」
0a      LF
```

それから、Shift-JIS。こちらは Windows 向きということで、CR+LF にしました。傾向は EUC に似ているけど、全角文字の 2 バイト目が 80 未満であることと、半角カナが 1 バイトなのが特徴ね。

```
--------|-0--1--2--3--4--5--6--7--8--9--A--B--C--D--E--F-|
00000000|82 50 82 51 31 32 82 50 82 51 0d 0a 83 4a 83 4b |
00000010|b6 de 0d 0a                                     |
--------|------------------------------------------------|
```

```
82 50   「１」
82 51   「２」
31      「1」
32      「2」
82 50   「１」
82 51   「２」
0d      CR
0a      LF
83 4a   「カ」
83 4b   「ガ」
b6      「ｶ」
de      「ﾞ」
0d      CR
0a      LF
```

第5回　JIS（ISO-2022-JP）の海へ

クリ：ななちゃん、また、同じ内容をエディタで入力して、今度は、JIS コード（ISO-2022-JP）でファイルに書き込んでから、それを「16 進ダンプ」で読み込んでみてくれんか。ついでに、解析も。

```
１２１２12
カガｶﾞ
```

なな：は～～い。これでいいかしら。モードを切り替える 3 バイトがあることと、すべてのバイトが 80 未満であるのが特徴ね。、

```
--------|-0--1--2--3--4--5--6--7--8--9--A--B--C--D--E--F-|
00000000|1b 24 42 23 31 23 32 1b 28 42 31 32 1b 24 42 23 |
00000010|31 23 32 1b 28 42 0d 0a 1b 24 42 25 2b 25 2c 1b |
00000020|28 49 36 5e 1b 28 42 0d 0a                      |
```

```
1b 24 42:     新 JIS 漢字モード
23 31:        「１」
23 32:        「２」
1b 28 42:     ASCII モード
31:           「1」
32:           「2」
1b 24 42:     新 JIS 漢字モード
23 31:        「１」
23 32:        「２」
1b 28 42:     ASCII モード
0d:           CR
0a:           LF
1b 24 42:     新 JIS 漢字モード
25 2b:        「カ」
25 2c:        「ガ」
1b 28 49:     半角カナモード
36:           「ｶ」
5e:           「ﾞ」
1b 28 42:     ASCII モード
0d:           CR
0a:           LF
```

クリ：よくできました。

なな：これだけいろいろな種類があると、どのコードを使っているかが分からないと、正しく解釈できないのが納得できるわ。UTF-8/16 みたいに、先頭にどのコードを使っているかを示すマーカがあれば良いけど。

クリ：先頭のマークは必須ではないんだよ。ウェブページのヘッダなどでコードが宣言されていなければ、ブラウザなどのソフトがデータの並び方をチェックして、一生懸命、どのコードを使っているか「推定」する必要があるんだ。推定に失敗すると文字化けが生じる。ユーザが、ソフトの設定で、どのコードであるかを陽に設定ことが必要になってくる。

第6回 Unicode とは

先生: ここから、私が説明します。Unicode は、世界中の文字を一括して登録することを目標にした文字コード体系です。

なな: 文字コード体系？

先生: するどいわね。文字コード体系というのはあいまいなことばなの。「文字集合」と「符号化方式」に分けて考える必要があります。

文字集合（Coded Character Set, CCS、符号化文字集合）は、表現したい**文字の範囲**のことです。半角アルファベット、半角カナ、全角文字（JIS第一水準、第二水準）など。いろいろな文字を集めて番号を振って管理したものです。Unicode は、この一種です。

これに対して、符号化方式 (Character Encoding Scheme, CES)、文字コード系（character encoding system）は、文字集合を構成する個々の文字の表現方法（数値の割り当て）です。集められた文字を、**コンピューターで使うとき**にどのような番号を振るか管理したもの。UTF-8、UTF-16 がこの例です。

Unicode は、UCS、Universal Coded character Set として、集合定義したもの。「Unicode スカラー値」という、U+ を付けた 4 桁から 6 桁の 16 進数で管理します。

- U+0000～U+007F までは ASCII の上位に 00 をつけ、U+ をつけたもの。たとえば、「 」（空白）はU+0020、!（半角の感嘆符）はU+0021、A（半角の大文字エー）はU+0041、a（半角の小文字エー）はU+0061。
- 日本語の全角文字 JIS X 0201 は、上位に FEC0 を付加し、先頭に U+ を付けたもの。
- 世界中の多くの文字は 16 進数 4 桁に収まっています。

- 1980 年代の当初の構想では、Unicode は 16 ビット固定長、2^{16} = 65,536 個の符号位置に必要な全ての文字を収録する、というもくろみでした。しかし、Unicode 1.0 公表後、拡張可能な空き領域 2 万字分を巡り、各国から文字追加要求が起こりました。中国、日本、台湾、ベトナム、シンガポールの追加漢字約 1 万 5 千字、古ハングル約 5 千字、未登録言語の文字など。Unicodeの、16 ビットの枠内に全世界の文字を収録するという計画は早々に破綻し、1996 年の Unicode 2.0 の時点で既に、文字集合の空間を 16 ビットより広げることが決まりました。
- U+10FFFF までの、1,114,111文字、約 100 万字までが収録できるようになりました。
- 現在、実際に使われているのは 2013 年 9 月 30 日に公開された Unicode 6.3.0で、110,187 文字です。

なな: Unicode は文字集合、UTF-8、UTF-16 はその文字符号化方式なのね。

先生: Unicode の文字符号化方式は、いくつかあります。普通にパソコンで使うのは、UTF-16 と UTF-8 の 2 種類です。UTF-16 には、さらに、ビッグ・エンディアン と リトル・エンディアンの 2 種類があります。UTF は Unicode Transformation Format（Unicode変換形式）の略。

まず、UTF-16（ビッグ・エンディアン）。

- ファイル先頭に 0xFEFF というコードを置く場合がありますが、必須ではありません。これは「幅ゼロの改行なしの空白」（Zero Width Non Breaking Space、ZWNBS）という見えない文字。Unicode のどのエンコーディング・スキームを使っているか、読み込むプログラムに伝えるためのものです。この用途で使う場合の ZWNBS のことをBOM（Byte Order Mark、バイト順マーク）と言います。正確に

は、BOM を付けないビッグ・エンディアンの UTF-16 は UTF-16BE と呼びます。

- U+FFFF までの Unicode スカラー値はそのままの16 進数 4 桁になります。U+10000以降の文字は、16進数 8 桁になります。サイズが倍になるので、「サロゲート・ペア」と呼びます。
- サロゲートペアは、16 ビット Unicode の領域 1024 文字分を 2 つ使います。 最初の16ビットユニットを high surrogate、二番目を low surrogate と呼びます。high surrogates は U+D800 〜 U+DBFF の範囲、low surrogates は U+DC00 〜 U+DFFF の範囲。1024 × 1024 = 1,048,576 文字を表す。第1面〜第16面（U+10000〜U+10FFFF）の文字をこれで表すこととした。これで、Unicode は合計で 1,048,576 + 65,536 - 2,048 = 111 万 2,064 文字ぶんの空間。

つぎに、UTF-16（リトル・エンディアン）。ファイル先頭に 0xFFFE というコードを置く。これは ZWNBS。Unicodeでは、U+FFFEという文字は絶対に定義されないことになっています。

なな: じゃあ、UTF-8 のほうは？

先生: ASCII の範囲では、ASCII と完全互換の 7 ビット。ASCIIに対して上位互換となっています。

- UTF-8 は 1 オクテット以上の可変長文字で、漢字や仮名などの表現に、主に 3バイトを使います。
- 規格上は6オクテットまで存在するが、現在4オクテットまでしか使われていなません。

U+0000 ……	U+007F	00-7F
U+0080 ……	U+07FF	C2-DF 80-BF
U+0800 ……	U+FFFF	E0-EF 80-BF 80-BF
U+10000 ……	U+1FFFFF	F0-F7 80-BF 80-BF 80-BF
U+200000 …	U+3FFFFFF	F8-FB 80-BF 80-BF 80-BF 80-BF
U+4000000 …	U+7FFFFFFF	FC-FD 80-BF 80-BF 80-BF 80-BF 80-BF

- ファイル先頭に EF BB BFというコードを置く。BOM。U+FEFFのUTF-8での表現。必須ではない。日本国内でのみ、BOM がついているものを UTF-8、ついていないものを UTF-8N として区別。

どのコードを使わなければならない、と決めてしまうと、古いデータを持っている人や、古いソフトを使っている人、作った人などが困るので、いろいろなコードが使われたままになっています。

全角文字が多い文書データでは UTF-16 のほうが、ファイルサイズが小さくなります。プログラムのように、ASCII 文字が多いデータでは、UTF-8 のほうが、ファイルサイズが小さくなります。

現在の主流は、ファイル保存は UTF-8、JavaScript の内部表現は Unicode スカラー値。日本語の文字を中心に考えれば、JavaScript の内部表現は UTF-16 と言ってもよいでしょう。HTML の <head> に、<meta charset="utf-8"> と書くのが定番です。

UTF-8 や UTF-16 は、世界共通で、日本語、中国語、韓国語などを混在させて使うことができます。JIS、Shift-JIS、EUC などは、日本語、中国語、韓国語など、言語ごとに異なり、混在させて使うことができません。

明朝体、ゴシック体、太字、斜体字、などの区別は、文字コードとは別の、「フォント」の仕様です。

第7回 文字コードの歴史

先生: 初期の、ASCII、ISO 8859-1、JIS X 0201 は、符号化文字集合でもあり、文字符号化方式でもありました。

ASCII は、英数字（A～Z、a～z、0～9）と一部の記号（!@#$%など）を含む文字集合で、区別するのに 7 ビット必要でした。00-7F。8 ビット目の最上位ビットは通信エラーを検出するためのパリティビットとして利用されました。たとえば、8 ビット全体での 1 の数が偶数個になるようにして送信し、受信時に偶数個でなければエラー発生と判断しました（偶数パリティ）。（奇数にするのが奇数パリティ。）

それを拡張した ISO 8859-1。ASCII の英数字に加え、フランス語のアクサンなどヨーロッパ特殊文字（À、0xC0など）、著作権記号（©、0xA9）のような記号が入れられました。8 ビットをフルで使う。00-FF。

日本で作られた JIS X 0201。ASCII の英数字に加え、半角カナが入っている。8 ビットをフルで使う。00-FF。ISO 8859-1で À が入っている C0 には ﾀ、© が入っている A9 には ｩ（ﾄｩｰｽなどの小文字のウ）が入っている。ISO 8859-1 と JIS X 0201 は競合するので、同時には使えない。

なな: このあと、全角文字が登場するのね。

先生: JISコード (ISO-2022-JP) は、エスケープシーケンスと呼ぶ特別な文字並びによりモード遷移を行うのが特徴。すべてのバイトが 7 ビット、つまり、最上位ビットをパリティとして使える。しかし、データ全体の先頭から追跡（スキャン）しないと解読できない。エスケープシーケンスに通信エラーがあると次のエスケープシーケンスまで解読不能になる。コンピュータ等によるデータ編集（データ削除や追加）が厄介。たとえば、漢字データブロックの中に ASCII 文字をはさむと、前後にエスケープシーケンスを挿入しないといけない。

- 00-1F、7F は制御コード
- エスケープシーケンス 文字データ

エスケープシーケンス		文字データ	
1B 28 42	[ESC (B]	20-7E	ASCII
1B 28 4A	[ESC (J]	20-7E	JISローマ字
1B 28 49	[ESC (I]	21-5F	JISカナ(半角カナ)
1B 24 40	[ESC $ @]	21-7E 21-7E	旧JIS漢字(1978)
1B 24 42	[ESC $ B]	21-7E 21-7E	新JIS漢字(1983/1990)
1B 24 44	[ESC $ D]	21-7E 21-7E	JIS補助漢字

なな: 全角文字が 7 ビットバイトの並びとして送れるけど、編集などのコンピュータ処理がやりにくいという問題があったのね。

先生: 日本語EUC (euc-jp) は、JIS漢字の文字集合を扱う文字符号化方式のひとつとみなせます。
Extended Unix Code の略。日本語 UNIX で使われているコード。
JIS漢字コードの上下バイトはともに最上位ビットがゼロであることに注目し、最上位ビットを 1 に変えることで EUC コードとする。つまり、最上位ビットが 0 のバイトはASCII文字、最上位ビットが 1 のバイトは 2 バイト連結し、最上位ビットを 0 に変えることで JIS漢字コードとして取り出せます。

- 00-1F、7F (1 byte) 制御コード
- 20-7E (1 byte) ASCII文字
- A1-FE A1-FE (2 byte) 全角文字（JIS X 0208の漢字エリア）
- 8F A1-FE A1-FE (3 byte) JIS補助漢字
- 8E A1-DF (2 byte) **半角カタカナ。未対応のシステムが多い。**

なな: 8 ビットバイトにして、エスケープシーケンスをなくしたので、データの一部を見るだけで半角か全角か分かるし、データ編集も容易にできるようになったわけね。じゃあ、Shift-JIS は？

先生: SJIS (Shift-JIS) も、JIS漢字の文字集合を扱う文字符号化方式のひとつとみなせます。
Microsoft 社が中心となって決めたコードで、Windows、Mac などで使用しています。半角カナは 1 バイト。**半角文字は 1 バイト、全角文字は 2 バイト。表示桁数と内部バイト数が一致するのが特徴**です。JIS 漢字のコードの 2 バイト目が ASCII と重なることや、JIS コードとの相互変換は計算処理で行えるけど、EUC よりはかなり複雑（複雑なシフト）なのが、問題といえば問題。

- [00-1F、7F] (1 byte) 制御コード
- [20-7E] (1 byte) JISローマ字(ASCII)
- [A1-DF] (1 byte) JISカナ(半角カナ)
- [81-9F/E0-EF] [40-7E/80-FC] (2 byte) JIS漢字

なな: エディタのファイル保存データのコード指定に ANSI というのも見たことがあるけど。

先生: ANSI は、American National Standards Institute（米国国家規格協会）の略。日本のJISにあたる国家規格を定める団体なの。でも、実は、Shift_JIS のこと。正確には、Shift_JIS をMicrosoftが拡張した CP932 のことなの。

なな: 米国国家規格協会なのに漢字のコードという、変なネーミングね。

注意: Shift-JIS には、歴史的には、仕様の細部が異なる亜種（CP932、shift_jis2000、shift_jis2004、x-mac-japanese など）がいろいろあった。使用頻度の少ない漢字や「外字」の扱いが異なる。

注意: Mac OSX では、ファイルは UTF-8、内部コードは Unicode(UTF-16)。

注意: HTML5 では、UTF-8 が基本。JavaScript の内部コードは Unicode (UTF-16)。

注意: Unicode（文字集合）と、UTF-16（符号化方式）は別物だが、半角文字、日本語の全角文字の範囲、つまり、U+FFFF 以下では、Unicode のスカラー値と、UTF-16 ビッグエンディアン のコード値は一致するので、曖昧のままで済まされているのが現実。

第8回 文字化けトリビア(1) 「 ¥ 」と「 \ 」

なな: トリビアって何？

先生: 英語で、trivia、くだらない、些細なことという意味よ。日本ではテレビ番組の影響で「雑学」という意味もあるわ。ところで、C 言語を始めとする多くのプログラミング言語では、文字列の中に、改行、タブ、「”」など、そのまま記述できない文字を扱うための「エスケープ」というものがあります。この印としてバックスラッシュ（「 \ 」）を使用しています。たとえば、改行を表す「 \n 」。それから、Windows のパス名の区切り文字としても使われています。

なな: それだったら円記号（「 ¥ 」） じゃないの？

先生: これは、有名な文字化けなのよ。有名すぎて、「 ¥ 」 のほうが本物だと誤解している人も多いくらい。

なな: どうして、そんな文字化けが起きるの？

先生: もともとアメリカで作成された ASCII 文字コード体系というのがあって、英数字や各種記号を 7bit の文字コードで表していたの。それが、国際規格、ISO 646 として採用されました。でも、国際規格とするには、ASCII はアメリカ使われる英数記号を対象としていたので、ヨーロッパやアジアでは不十分だった。たとえば、「Á」、「Ä」、「ç」 がアルファベットに含まれる言語がある。しかも、それぞれの国によって事情が異なる。そこで ISO 646 では、ASCII 文字のうち 12 の記号に関しては各国の必要に応じて変更できるという規格になった。

なな: 12 の記号って？

先生: # $ @ [\ ] ^ ` { | } ~ の 12 個よ。そして、この ISO 646 をもとに、日本で制定されたのが JIS X 0201 という規格。

なな: 半角カタカナが含まれるヤツ？

先生: そうね。ASCII や ISO 646 は 7 bit（00_{16}~$7F_{16}$） のコードなんだけど、JIS X 0201 は ISO 646 互換の 7 bit 領域に加え、8 bit（80_{16}-FF_{16}） 領域も使って、半角カタカナ、句読点、かぎ括弧などを定義したの。$E0_{16}$ から FE_{16} までは将来のために使わず、$A0_{16}$ は 20_{16} と同じ空白文字にしました。

なな: ISO 646 互換の 7 bit 領域の、各国用 12 文字はどうしたの。

先生: 10 文字は、ASCII と同じだけど、2 文字だけ取り換えました。「 \ 」→「 ¥ 」 と、「 ~ 」（チルダ）→「 ‾ 」（オーバーライン）。

なな: 「 ¥ 」 と「 \ 」 の登場ね。「 ¥ 」 を入れたかったけど、「 $ 」 は日本語でも使うけど、「 \ 」 は使うことが少ないと考えたのね。日本語キーボードでは、英語（インターナショナル）仕様のキーボードで \ の位置に¥ と刻印されているわね。

先生: まあ、今では、コンピュータで全角文字が自由に使えるので、「 ¥ 」 と「 \ 」 は同じ文字だと思ってしまえば、実用上は困らないわね。全角文字の「￥」と「＼」 は別の文字だし。

なな: 日本以外は、ISO 646 の 12 文字をどうしているの？

先生: 下記のようになっています。

国名	23	24	40	5B	5C	5D	5E	60	7B	7C	7D	7E
アメリカ	#	$	@	[	\	]	^	`	{	|	}	~
日本	#	$	@	[	¥	]	^	`	{	|	}	‾
イギリス	£	$	@	[	\	]	^	`	{	|	}	~
フランス	£	$	Á	°	Ç	§	^	µ	é	ù	è	¨
ドイツ	#	$	§	Ä	Ö	Ü	^	`	ä	ö	ü	ß
スウェーデン	#	¤	É	Ä	Ö	Å	Ü	é	ä	ö	å	ü
スペイン	#	$	·	¡	Ñ	Ç	¿	`	´	ñ	ç	¨

なな: たとえば、ドイツでは、エスケープ文字やパス名の区切り文字として「Ö」を使っているの？

先生: それではあんまりだということか、ヨーロッパでは DOS Latin 1（CP850） というコードを使っているようです。ISO 646 の部分は ASCII 文字をそのまま使い、80_{16} 部分にヨーロッパ言語用半角文字をあてはめています。ヨーロッパ全体をカバーできるような、広範な文字群が定義されています。

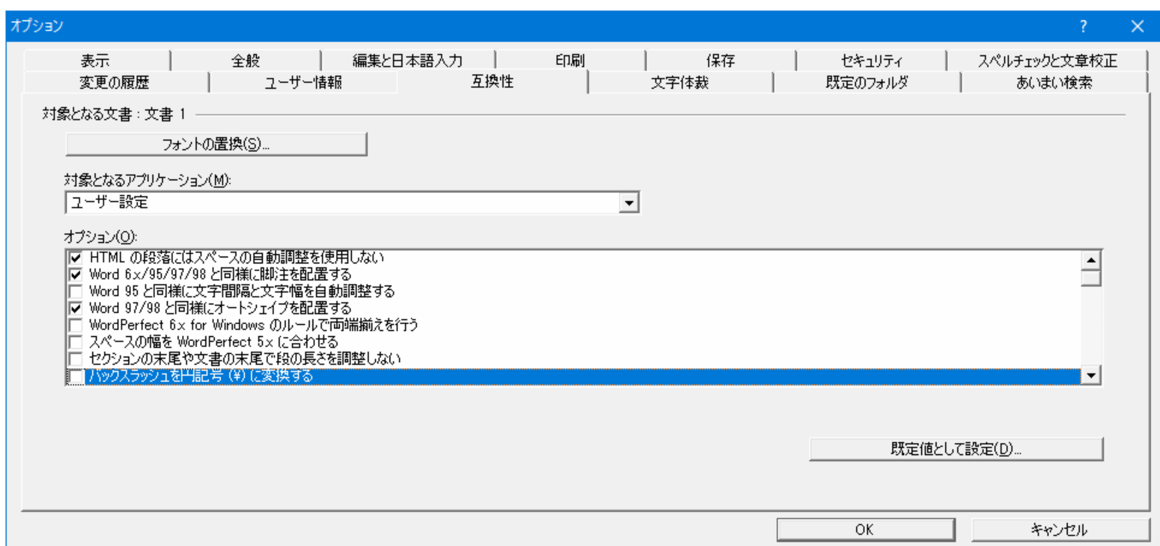
CP850 (DOS Latin 1)

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	[ ]	[☺]	[☻]	[♥]	[♦]	[♣]	[♠]	[●]	[◘]	[○]	[◙]	[♂]	[♀]	[♪]	[♫]	[☼]
10	[►]	[◄]	[↕]	[‼]	[¶]	[§]	[▬]	[↨]	[↑]	[↓]	[→]	[←]	[∟]	[↔]	[▲]	[▼]
20		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
80	Ç	ü	é	â	ä	à	å	ç	ê	ë	è	ï	î	ì	Ä	Å
90	É	æ	Æ	ô	ö	ò	û	ù	ÿ	Ö	Ü	ø	£	Ø	×	ƒ
A0	á	í	ó	ú	ñ	Ñ	ª	º	¿	®	¬	½	¼	¡	«	»
B0	░	▒	▓	│	┤	Á	Â	À	©	╣	║	╗	╝	¢	¥	┐
C0	└	┴	┬	├	─	┼	ã	Ã	╚	╔	╩	╦	╠	═	╬	¤
D0	ð	Ð	Ê	Ë	È	ı	Í	Î	Ï	┘	┌	█	▄	¦	Ì	▀
E0	Ó	ß	Ô	Ò	õ	Õ	µ	þ	Þ	Ú	Û	Ù	ý	Ý	¯	´
F0	−	±	‗	¾	¶	§	÷	¸	°	¨	·	¹	³	²	■	[?]

ASCII
互換
部分

なな: MS Word で「 \ 」を入力しようとすると、「 ¥ 」に変わってしまうんだけど。

先生: オプションメニューで、「バックスラッシュを円記号(¥)に変換する」 がデフォルト(初期値)で選択されているからね。これを非選択に変えないと、\ と入力しても ¥ に変えられてしまうのよ。



MS IME では、ひらがなモードで、U00AF のような形で Unicode を入力し、F5 キーを押すと候補一覧が表示されるので、マウスで選択し、Enter キー入力すれば大丈夫。Unicode では、U+005C がバックスラッシュ、U+00A5 が円記号として区別されるからね。

なな: MS-DOS は、なぜパス区切り文字としてバックスラッシュ 「 \ 」を採用したの? UNIX のように「 / 」を使えば良いのに。スラッシュはもともと日付や数字などを区切るのに用いられる記号なので、パスの区切りにふさわしいと思うけど。「 / 」なら日本語環境でも化けないし。

先生: 「 / 」は使えない事情があったのね。MS-DOS には、はじめ、様階層型ファイルシステムはなく、「パス区切り」という概念自体がありませんでした。MS-DOS 2.0 から階層構造が取り入れられたのね。でも、パソコンでは、CP/M の時代から、伝統的に、コマンドラインオプションを表す記号としてスラッシュを使用していたの。UNIX の 「% ls **–l** 」の形に対し、「>dir **/A** 」のようにね。オプション記号とパス区切りが同じだと問題を引き起こす可能性があるため、パス区切りにスラッシュを使うことができなかったの。そこで、ほぼ同じ形のバックスラッシュを使用したということね。

なな: 日本語キーボードに 「 \ 」にないのはどうしようもないけど、UTF-8 や UTF-16 では、文字コードの上では 「 \ 」と 「 ¥ 」をちゃんと区別できるのね。よかった。

第9回 文字化けトリビア(2) Shift_JIS コードの悲劇

なな: Shift-JIS コードの悲劇って何？

先生: 日本語 UNIX で用いられる EUC コードでは、全角文字は、80_{16} 以上のふたつのバイトで構成されているのね。ASCII 半角文字は $7F_{16}$ 以下の 1 バイト。だから、80_{16} 以上のコードの最上位ビットを破壊しない限り、プログラムがそのまま使えることが多いの。

なな: ところが、Shift_JIS (SJIS, cp932) の文字コードでは、2 バイト目が $7F_{16}$ 以下になるものが多々あるわね。

先生: そう。$7F_{16}$ 以下のバイトデータがあったとき、直前のバイトが 80_{16} 以上（全角文字の 2 バイト目）なのか、そうでない（半角文字）のかを区別しないといけないの。 全角文字の 2 バイト目で、半角文字のコードと重なる文字に、@ [\ ] ^ _ ` { | } ~ があるの。半角文字用に作られたプログラムをそのまま使うと誤動作する原因になるのね。その代表選手は「 \ 」($5C_{16}$)。「 \ 」は、もともと、日本語環境で円記号（「 ￥ 」）として表示されることがある問題の文字（トリビア(1)参照）。MS-DOS や Windows で、パス名の区切り文字に使ったり、一般にプログラミング言語の文字列内で改行コードなどを表現するのに使う「エスケープ文字」として使われる文字。2 バイト目が $5C_{16}$ になる全角文字には「―」、「ソ」、「十」、「表」など、使用頻度の高いものもあるの。

― ソ Ы Ⅸ 噂 浬 欺 圭 構 蚕 十 申 曾 箪 貼 能 表 暴 予 禄 兔
喀 媾 彌 拿 杤 歃 濬 畚 秉 綵 臀 藹 觸 軆 鐔 饅 鷭 纊 犾 偆 砡

2 バイト目が $7C_{16}$ のパイプ文字（ | ）になっている文字も、問題を引き起こす可能性がある。

ー ポ л 榎 掛 弓 芸 鋼 旨 楯 酢 掃 竹 倒 培 怖 翻 慾 處 嘶
孛 忿 掟 桍 毫 烟 痞 窩 縹 艚 蛞 諫 轎 閖 驂 黥 僴 礰 埈 蒴

たとえば、プログラミング言語に関連するコンパイラやインタプリタのプログラムなどで、全角文字に対応していないものは、これらの全角文字を半角文字と誤認して、変な動作になることがあるの。たとえば、「 “ 」で囲まれた文字列の中にあっても、次の文字をエスケープしてしまったり、文字列が途中で終了しているとみなされてしまったり。

なな: それも、広い意味で「文字化け」ということね。

先生: そうね。でも、Shift-JIS をちゃんと意識したプログラムを使えば大丈夫だし、最近はプログラムもデータも UTF-8 を使うことを前提とすることが多くなっているから問題はほとんどないわ。半角文字前提のプログラムに、Shift-JIS のデータを読ませて誤動作する時、データを EUC や UTF-8に変換してから使うと正常動作する場合がある、ということだけ覚えておくと、得することがあるかも。

なな: わかりました。

おまけ: JavaScript(ブラウザ)を使って、文字コードを調べる

文字を Unicode 16 進数表記に変換する。

```
<script>
alert("あ".charCodeAt(0).toString(16));
</script>
```

16 進数表記の Unicode を文字に変える。

```
<script>
alert(String.fromCharCode(0x3042));
</script>
```

対話型のプログラム。

文字コード確認

文字 あ → 16 進 3042

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="utf-8">
      <title> 文字コード確認</title>
      <style> input { font-size:150%; }</style>
   </head>
   <body>
     <h2>文字コード確認</h2>
     文字<input type=text id=dec value="あ" style="text-align:left;" onchange=right() size=4>
     <input type=text id=dir value="→" readonly onclick=chdir() size=1>
     <input type=text id=rdx value="16" size=2   onchange=recalc()>進
     <input type=text id=tgt value="3042" style="text-align: right;" onchange=left()>
<script>
var decp = document.getElementById("dec");
var tgtp = document.getElementById("tgt");
var dirp = document.getElementById("dir");
var rdxp = document.getElementById("rdx");
var decp, tgtp, dirp, rdxp;

function chCode(s) { return s.charCodeAt(0); }
function toChar(code) { return String.fromCharCode(code); }
function right() {
    dirp.value = "→";
    tgtp.value = chCode(decp.value).toString(rdxp.value);
function left() {
    dirp.value = "←";
    decp.value = toChar(parseInt(tgtp.value,rdxp.value)); }
function chdir() {
    if(dirp.value == "←") {
        dirp.value = "→";
        tgtp.value = chCode(decp.value).toString(rdxp.value);
    } else {
        dirp.value = "←";
        decp.value = toChar(parseInt(tgtp.value,rdxp.value)); }   }
function recalc() {
    if(dirp.value == "→") {
        tgtp.value = chCode(decp.value).toString(rdxp.value);
    } else {
        decp.value = toChar(parseInt(tgtp.value,rdxp.value)); }   }
</script>
</body>
</html>
```