■#10_文字コードのお話

今回、“コンピュータネットワーク” について考える前に、あらためて文字コード について考えてみます。

1. 文字コードとは

コンピュータで通信を行う手段のひとつとして、前回 “シリアル” と “パラレル” 通信について勉強しました。今日はその通信の中で行われているデータについて考えてみましょう。

2. コンピュータ通信で扱うデータ

コンピュータが扱えるデータは基本的に “0：LOW” 、 “1：HIGHH” の 2 種類しか無いことを勉強してきました。その ”0“、”1“ を組み合わせ、人間が利用できるデータとして利用しており、ひとつ々の情報を “bit:ビット” 、ビットを 8 個組み合わせて “Byte：バイト” とすることは理解されていると思います。

人間が情報として用いる場合、英数や漢字、ひらがな、カタカナなどの文字を利用しています。通信でも “文字” データを利用することが多く、これら文章として扱うデータを “テキストデータ” と呼びます。

そのテキストデータとして、ミニマムなものとしては、数字：0～9、 大文字： A～Z 、小文字：a～z の合計 62 文字となります。この場合 2^6 あれば 1 文字を表現できます。しかし文章には英数文字だけでは無く、ピリオドやクオーテーションマークなどの記号や、改行などの制御コードも必要ですので、余裕を見て 7bit で文字を表現することにしました。これが “ASCII コード” と呼ばれるものになり、文字コードの基礎になります。

GC2018 コンピュータ概論 Ⅱ

ASCII コード表

10 進	16 進	文字･記号･制御コード
制御コード		
0	0x00	**NUL（null 文字）**
1	0x01	SOH（ヘッダ開始）
2	0x02	STX（テキスト開始）
3	0x03	ETX（テキスト終了）
4	0x04	EOT（転送終了）
5	0x05	**ENQ（照会）**
6	0x06	**ACK（受信確認）**
7	0x07	BEL（警告）
8	0x08	BS（後退）
9	0x09	HT（水平タブ）
10	0x0a	**LF（改行）**
11	0x0b	VT（垂直タブ）
12	0x0c	FF（改頁）
13	0x0d	**CR（復帰）**
14	0x0e	SO（シフトアウト）
15	0x0f	SI（シフトイン）
16	0x10	DLE（データリンクエスケープ）
17	0x11	DC1（装置制御１）
18	0x12	DC2（装置制御２）
19	0x13	DC3（装置制御３）
20	0x14	DC4（装置制御４）
21	0x15	**NAK（受信失敗）**
22	0x16	**SYN（同期）**
23	0x17	ETB（転送ブロック終了）
24	0x18	CAN（キャンセル）
25	0x19	EM（メディア終了）
26	0x1a	SUB（置換）
27	0x1b	**ESC（エスケープ）**
28	0x1c	FS（フォーム区切り）
29	0x1d	GS（グループ区切り）
30	0x1e	RS（レコード区切り）
31	0x1f	US（ユニット区切り）
記号		
32	0x20	SPC（空白文字）
33	0x21	!
34	0x22	"
35	0x23	#
36	0x24	$
37	0x25	%
38	0x26	&
39	0x27	'
40	0x28	(
41	0x29	)
42	0x2a	*
43	0x2b	+
44	0x2c	,

10 進	16 進	文字
45	0x2d	-
46	0x2e	.
47	0x2f	/
数字(NUM)		
48	**0x30**	**0**
49	0x31	1
50	0x32	2
51	0x33	3
52	0x34	4
53	0x35	5
54	0x36	6
55	0x37	7
56	0x38	8
57	0x39	9
記号		
58	0x3a	:
59	0x3b	;
60	0x3c	<
61	0x3d	=
62	0x3e	>
63	0x3f	?
64	0x40	@
大文字(AL)		
65	**0x41**	**A**
66	0x42	B
67	0x43	C
68	0x44	D
69	0x45	E
70	0x46	F
71	0x47	G
72	0x48	H
73	0x49	I
74	0x4a	J
75	0x4b	K
76	0x4c	L
77	0x4d	M
78	0x4e	N
79	0x4f	O
80	0x50	P
81	0x51	Q
82	0x52	R
83	0x53	S
84	0x54	T
85	0x55	U
86	0x56	V
87	0x57	W
88	0x58	X

10 進	16 進	文字
89	0x59	Y
90	0x5a	Z
記号		
91	0x5b	[
92	0x5c	¥
93	0x5d	]
94	0x5e	^
95	0x5f	_
96	0x60	`
小文字（AL）		
97	**0x61**	**a**
98	0x62	b
99	0x63	c
100	0x64	d
101	0x65	e
102	0x66	f
103	0x67	g
104	0x68	h
105	0x69	i
106	0x6a	j
107	0x6b	k
108	0x6c	l
109	0x6d	m
110	0x6e	n
111	0x6f	o
112	0x70	p
113	0x71	q
114	0x72	r
115	0x73	s
116	0x74	t
117	0x75	u
118	0x76	v
119	0x77	w
120	0x78	x
121	0x79	y
122	0x7a	z
記号		
123	0x7b	{
124	0x7c	|
125	0x7d	}
126	0x7e	~
制御コード		
127	0x7f	**DEL**

3. 日本語を表現する文字コード。#01 日本にかけられた呪い Shift_JIS

これで、英数文字を表現することが出来ましたが、日本語はそのほかにも、ひらがな、カタカナ、漢字があります。

そしてひらがな、カタカナにはそれぞれ、“だ、が” のような濁音 “ぱ、ぴ” のような半濁音、“きゃ、ひゃ” などの拗音、“きっ、ぱっ” などの促音、さらに “だー、やー” のような長音があります。

また、漢字は、現在文字コードの対象になっているものだけで、第一水準 2,965 字、第二水準 3,390 字、第三水準 1,259 字、第四水準 2,436 字、非漢字 1,183 字、合計 11,233 字があります。

どうしましょう？

日本語は必要な文字数が多く、古い時代では漢字の利用が出来ず、印刷しやすい カタカナ のみ利用されていました。そのカタカナをどこに入れたかというと、ASCII では 7bit(128 文字)を利用していたので、1bit 追加し 8bit(256 文字)として、空いた隙間にカタカナのコードを押し込んでいます。
また、濁音や半濁音は “゛”、“゜” と独立した文字として準備されていました。

0x00～0x1f、0x7f が 制御コード
0x20～0x7e が ASCII 文字(英数文字、記号)

↓追加部分
0x21～0x5f が JIS7(7 ビット JIS)の半角カタカナ ※切り替えて利用。現在ほとんど利用されていない。
0xa1～0xdf が JIS8(8 ビット JIS)の半角カタカナ

漢字は 2Byte を使って表現する。
第 1Byte 0x80～0x9f、0xe0～0xef
第 2Byte 0x40～0x7e、0x80～0xfc ※ASCII 制御文字に重複しないよう設定

後に、漢字を利用しやすくするためマイクロソフトや他メーカにより、”シフト JIS” と呼ばれるコードに拡張されて、日本ではこれがスタンダードな文字コードである時期が長く続きました。

3.1. “シフト JIS” の問題点。

ASCII コードを利用して、日本語を表現するために、”シフト JIS” では、それまでの “JIS コード” では別なコードとして扱われていた、”英数文字” と ”漢字” のコードをひとつの文字コードで表現したため、様々な問題を持っています。

参考：https://ja.wikipedia.org/wiki/Shift_JIS

3.1.1. 複雑なコード体系

文字コードをずらす(シフト)ことで、単一のコード体系で英数、仮名、漢字を表現するため、コード自体が複雑になっています。

3.1.2. バックスラッシュ ”¥” の呪い

一般的に UNIX 系の OSS では PATH の区切りに “/” を利用するため特に問題は無いのですが、MS-DOS に実装する際、お手本にした CP/M ですでにオプションの指定に利用されていたため、MS-DOS ではよく似た ”¥” を利用していたことで悲劇が始まりました、というか呪いレベルに近いです。

実は ASCII コードは各国の都合に応じて、使用頻度が少なく、影響が少ないだろうと考えられている 12 文字を変更してもよいというルールがあり、日本では ASCII コードの中で利用頻度が少ないであろう、次の 2 つだけ変更が加えました。

それがチルダをオーバーラインにしたことと、バックスラッシュを円記号にしたことです。

そのため、日本では、Windows になっても、PATH 区切りは“￥”になってしまいました。

この文字化け現象は現代まで引き継がれており、Windows10 になり、OS 内部で使われる文字コードがほぼ Unicode になっており、もちろんバックスラッシュも円記号も区別できるのですが、どちらかというと

「今までと同じように見えた方がいい」

ということで、日本の Windows 上などでは、あいかわらず円記号で表示されています。

3.1.3. “0x5C 問題“

シフト JIS では、漢字を表現するために 2Byte を利用します。2Byte 目で利用される情報は、無用なトラブルを避けるため、制御コードを避けて設定されています。しかし、”0x5c”・・・実は 3.1.2 でも話した “0x5c：¥” が利用範囲に含まれています。

“0x5c” で表現される文字は、多くのプログラミング言語（C、Perl、Bourne Shell など多数）ではエスケープ文字としているため、マルチバイト非対応の環境では文字化けを引き起こす原因になります。

構		わ		な		い	
8d	**5c**	82	ed	82	c8	82	a2

▼バックスラッシュにあたる **5c** が抜けた場合

8d		82	ed	82	c8	82	a2
高			堋		ﾈ	い	

4. 日本語を表現する文字コード。#02 UNIX でのスタンダードだった EUC-JP

Extended Unix Code(EUC)は、UNIX 上で使われてきた文字コードの符号化方式で、EUC 上で日本語を扱えるようにしたものが、EUC-JP（Extended UNIX Code Packed Format for Japanese、日本語 EUC）です。

半角カタカナを利用するために、2Byte 消費するなどシフト JIS と比較して扱いづらい点も多少ありますが、文字表現のコード体系はシンプルで、英数字と日本語文字の区別がしやすく、プログラム上での扱いが楽な文字コード体系になり、UNIX 互換 OS の Linux、BSD では現在でも利用されています。

5. 日本語を表現する! #03 ワールドワイドな、Unicode(ユニコード)

一般の文字コードが、特定の言語を表記するために作られたものであるのに対して、世界中のすべての言語を単一の文字コードで表現することを目的に作られている文字集合の規格となります。

Unicode をコンピュータに実装する具体的な規格に、UTF-7、UTF-8、UTF-16、UTF-32 があります。

5.1. UTF-7 ：通信に合わせて決めてみたが・・・

Unicode を、旧 SMTP のような 7bit コードしか扱えない環境に向けて作られた規格。

しかし問題も多く、現在では UTF-8 を Base64 などで変換するなどの方式が推奨されています。

5.2. UTF-16 ：世界中の言葉を 2Byte に・・・

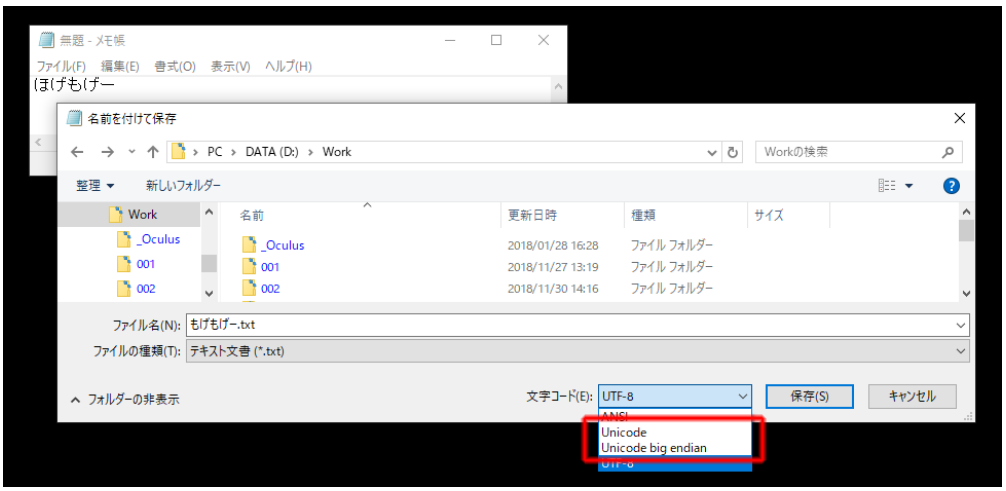
同時期に策定されていた ISO/IEC10646 を乗っ取る形で、Unicode を実質固定長 2Byte で表現しようとした規格。

2Byte で 1 文字を表現するため。バイトの順番が問題となり、ファイル先頭にバイト順の指定を行うか、リトルエンディアン(UTF-16LE)、ビッグエンディアン(UTF-16BE)など指定しなければ、Windows-スマホ など異なる環境での文字化けし通信ができない可能性があったり、結局 2Byte じゃだめだったりと。

また、一般的なアプリケーションで “Unicode で保存” という表現はおかしいのだが、古いアプリなどである場合大概 “UTF-16” のことです。

Ex) Windows のメモ帳とか・・・ Windows は大抵 IntelCPU なのでリトルエンディアン、スマホでよく利用される ARM は両刀遣いのバイエンディアンと、通信すると地味に響く

PC 同士だと問題なかった通信が、スマホと通信するとまともに動かないとか・・・



5.3. UTF-8 ： 可変長で柔軟に！ついでに互換性も！ 多分今のスタンダード

UTF-16 で固定長の運用に限界があり、可変長（1-4 バイト）の 8 ビット符号単位で表現するように作られた規格。

ASCII に対して上位互換となっており、文字の境界が明確なので、UTF-16 や UTF-32 との変換・逆変換に際して乗除算などの高負荷処理が必要ない、などの特長を持ち、インターネットではもっとも一般的に利用されている規格になります。

今のところ、迷ったら UTF-8 位で考えていて大丈夫と思います。

5.4. UTF-32 ：理想型？・・・ただ々でかすぎ

Unicode のすべての符号位置を単一長の符号単位として 32 ビットで表現しようとした規格。

でも実は、32bit すべてを利用せず。Unicode の符号空間が U+10FFFF までであるため 21bit しか使っていなかったりする。

非常に単純ですが、1 文字表現するために 4Byte も消費するため、保存形式として現在必要性が少ないため利用されることは少ない。

ただ文字変換処理などで、一時的に内部処理で利用されることがある。

※余談ですが、乗っ取る前の ISO/IEC10646 では 21bit 以上の空間も利用できるはずだった。

ほかにも、いろいろあるのですが、今回は割愛します。

6. 文字化け

ここにあげたものだけでも一部なのですが、世の中には新旧、各国や団体の思惑などで文字コードの世界はまあまあカオスです。

当然異なる文字コード間では、文字として認識できなくなることも多く、近年インターネットなど利用環境と異なる環境との情報のやりとりを、行う場合 「くぁｗせ drftgy ふじこ lp」 的な ？？？な状態になります。

文字化けの例

「おはよう abcd123」 を 文字コード変えて“UTF-8”で読み込んでみる

Shift-JIS → ｢u｢｢~窓 abcd123｢v

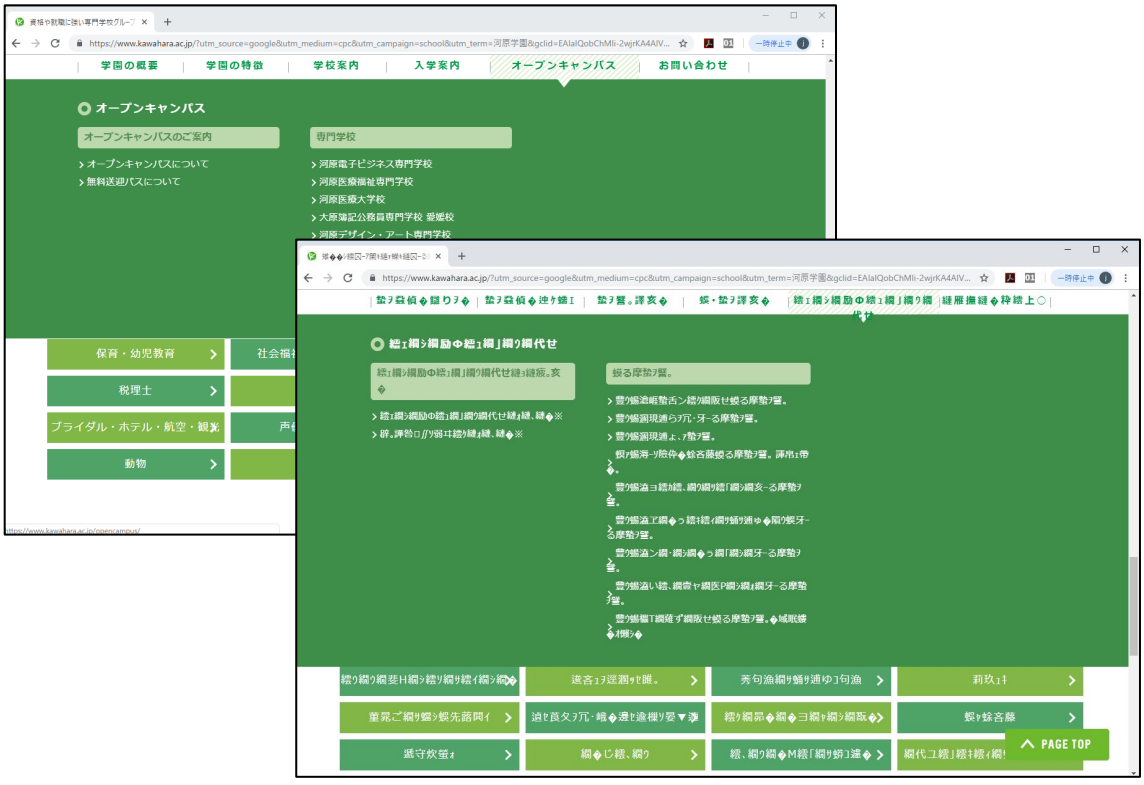
EUC-JP → ｢｢｢ｸ 襦 abcd123｢｢

かろうじて、互換性のある ASCII の英数文字が判別できるが、日本語は壊滅的な状態です。

6.1. インターネットはカオスです：当てにならない設定

インターネットの世界になって、実はこの文字コードの問題、かなり根深い問題になってきました。

ちなみに、河原学園のページを、意図的に文字コードを、設定された UTF-8 ではなく、SHIFT-JIS で読み込んだものが下記の通りになります。



これ、設定すれば大丈夫じゃ無いかと思われますが、途中のサーバで文字コードが勝手に変更されたりすることもたまにあり、“設定している文字コード” ＝ “実データ” があまり信用できなかったりします。

まあ、設定したつもりで、全然別の文字コードで保存される方もいますので・・・まあまあ多い、んでツボはまる。

「僕は悪くない！ちゃんと設定したんだから～」

本人の自覚がないので、Web デザインの世界や、データベースの世界などではかなり面倒な事態になったりします。

6.2. インターネットはカオスです：兄化け？何それ

2013 年のこと、Docomo がやっと iPhone の販売にこぎ着けました。そのとき、文字コード由来のちょいとした通信障害が起こって話題になったのです。

それまで、Docomo では iPhone が発売されて無かったため、Shift-JIS を独自拡張した CP932 という文字コードを使っていても問題なかったのですが、au、SoftBank の iPhone から送信されたメッセージを Docomo の iPhone が理解できず、絵文字がことごとく文字化けしたのです。

まあ、ビジネスでは大きな問題は無かったのですが、大事な絵文字が出せないと話題になり。実際に、兄から妹に来たメールで文字化けが出たことで 「兄化け」と呼ばれるようになったそうです。

参考：https://togetter.com/li/580702

7. 文字コード、少しでもいいので興味持ってくださいね

Web が一般的になった現代でも。“過去の遺物” 的な古いエンコードのデータが世界各地に埋蔵されています。ゲームでも様々な機器との通信を行う場合も増えてきていますので、少しずつでもいいので、文字コードに興味持ってください。

GC2018 コンピュータ概論Ⅱ

参考資料：興味ある人はこんなの見ると面白いかも

○ Unicode Wikipedia　https://ja.wikipedia.org/wiki/Unicode

○ Shift-JIS Wikipedia　https://ja.wikipedia.org/wiki/Shift_JIS

○ 学校では教えてくれないこと エンディアンの話

https://www.uquest.co.jp/embedded/learning/lecture05.html

○文字化け、Google 検索

https://www.google.com/search?client=firefox-b&ei=4o9GXIHLDdijwAPR2qOwDw&q=%E6%96%87%E5%AD%97%E5%8C%96%E3%81%91

○チェックポイント・キーワード

・文字コード
・ASCII
・Shift-JIS
・EUC-JP
・UTF-8
・文字化け

図は かわいいフリー素材いらすとや より引用