今回は、文字データの表現方法について学ぶ。

1. 原理

   文字をコンピュータで表現するためには、１年生で学んだ ASCII コード表のような表を用いる。ASCII コード表は英字、数字、記号しか含んでいなかったので、日本語や世界各国の文字を表すには不十分である。そこで、ASCII コード表の他に様々な文字コード表が用いられる。次の手順で文字コード表を決める。

   (a) 扱う文字の範囲を決める

   文字コード表に入れる文字の範囲を決める必要がある。決めた範囲に含まれる文字の集合を**文字集合 (character set)** と呼ぶ。

   (b) 文字に番号を付ける

   文字集合に含まれる全ての文字に番号を付ける。番号付けされた文字の集合を**符号化文字集合 (coded character set)** と呼ぶ。

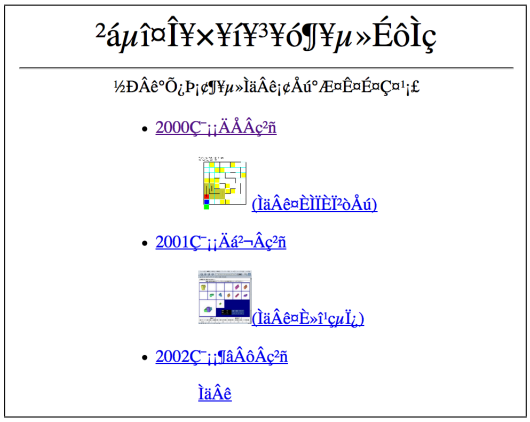
   (c) 番号をビット列に変換する方法を決める。

   文字に付けた番号と実際のビット（バイト）列との変換方法を決める。この変換方法のことを**文字符号化方式 (character encoding scheme)** と呼ぶ。よく使う文字が短いビット列になるように符号化したりする。複数の文字集合を同時に使用する時は、どの文字集合に属するか分かるように符号化方式を決める。

   どのコンピュータでも同じ文字データが扱えるように、符号化文字集合、文字符号化方式は ISO[1]、JIS[2]等で規格化されている。しかし、国により、コンピュータの OS により、アプリケーションプログラムにより、採用している規格が完全に同じではないため、どの規格か判定する際にミスがおこることがある。**文字化け**は、判定ミスが起きた結果である。

   Web ブラウザが文字符号化方式の自動判定に成功した例と失敗した例を図 1 に示す。

(a) 正しく判定できた



(b) 正しく判定できなかった

図 1: 文字化けの例

[1]ISO：International Organization for Standardization(国際標準化機構)
[2]JIS：Japan Industorial Standards(日本工業規格)

2. 日本で使用される符号化文字集合

(a) ASCII(American Standard Code for Infomation Interchange) 文字コード表

図 2 に示す符号化文字集合 (coded character set) のこと。1963 年にアメリカ規格協会 (ASCII) が定めた。7 ビットで、制御文字、記号、数字、英字大文字、英字小文字を表現できる。制御文字と印刷可能文字あわせて 128 文字が含まれる。

(上位3ビット)

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

印刷可能文字

*TAB: Tabulator(表作成) **機種やソフトによる

	意味	C言語	キー入力
BEL(07H)	ベル(Bell)	’\007’	^G
BS (08H)	バックスペース(Back Space)	’\b’	^H [BackSpace]
HT (09H)	タブ(Horizontal TAB)*	’\t’	^I [Tab]
LF (0AH)	改行(Line Feed)	’\n’	^J [Enter]**
FF (0CH)	改ページ(Form Feed)	’\f’	^L
CR (0DH)	復帰(Carriage Return)	’\r’	^M [Enter]**
ESC(1BH)	エスケープ(Escape)	’\033’	[Esc]
(20H)	空白(White Space)	’ ’	[ ]

代表的な制御文字

ファイル	通信回線：CR LF
Windows : CR LF UNIX : LF MacOS X : LF	プログラム中 ：LF(’\n’)

改行の表現方法

図 2: ASCII 文字コード表

(b) JIS X 0201(通称 JIS 8 ビットコード)

図 3 に示す符号化文字集合のこと。1969 年に日本工業規格 (JIS) に定められた。8 ビットで、制御文字、記号、数字、英字大文字、英字小文字、カタカナを表現できる。制御文字と印刷可能文字あわせて 191 文字が含まれる。制御文字、記号、数字、英字大文字、英字小文字の範囲は、ASCII と、ほぼ、同じになっている。

(上位4ビット)

(下位4ビット)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE		0	@	P	`	p				ー	タ	ミ		
1	SOH	DC1	!	1	A	Q	a	q			▒	ア	チ	ム		
2	STX	DC2	”	2	B	R	b	r			▒	イ	ツ	メ		
3	ETX	DC3	#	3	C	S	c	s			▒	ウ	テ	モ		
4	EOT	DC4	$	4	D	T	d	t			▒	エ	ト	ヤ		
5	ENQ	NAK	%	5	E	U	e	u			▒	オ	ナ	ユ		
6	ACK	SYN	&	6	F	V	f	v			ヲ	カ	ニ	ヨ		
7	BEL	ETB	’	7	G	W	g	w			ァ	キ	ヌ	ラ		
8	BS	CAN	(	8	H	X	h	x			ィ	ク	ネ	リ		
9	HT	EM	)	9	I	Y	i	y			ゥ	ケ	ノ	ル		
A	LF	SUB	*	:	J	Z	j	z			ェ	コ	ハ	レ		
B	VT	ESC	+	;	K	[	k	{			ォ	サ	ヒ	ロ		
C	FF	FS	,	<	L	¥	l	|			ャ	シ	フ	ワ		
D	CR	GS	−	=	M	]	m	}			ュ	ス	ヘ	ン		
E	SO	RS	.	>	N	^	n	▒			ョ	セ	ホ	゛		
F	SI	US	/	?	O	_	o	DEL			ッ	ソ	マ	゜		

▒ はASCIIコード表と異なる部分

図 3: JIS 8 ビット文字コード表

(c) JIS X 0208(通称 JIS 漢字)

図 4 に示す符号化文字集合のこと。1978 年に日本工業規格 (JIS) に定められた。16 ビットで、記号、英数字、かな、漢字などを表現できる。**全角文字** 6,879 文字が登録されている。

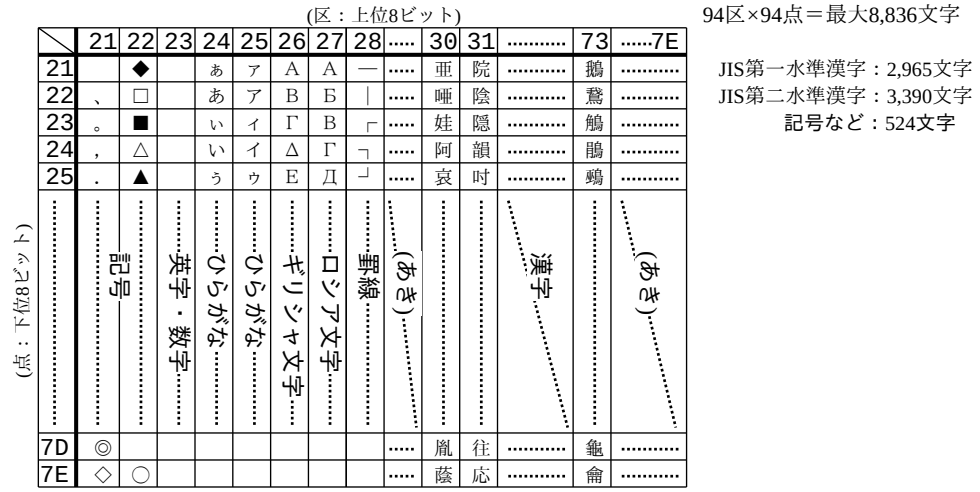


図 4: JIS 漢字コード表

(d) JIS X 0213(通称 JIS 拡張漢字)

図 5 に示す符号化文字集合のこと。2000 年に日本工業規格 (JIS) に定められた。その後、2004 年、2012 年に改正されている。コード表を２枚にすることで、より多くの文字を格納する。**全角文字** 11,223 文字が登録されている。JIS X 0208 を包含する上位規格 (super set, 上位集合) である。

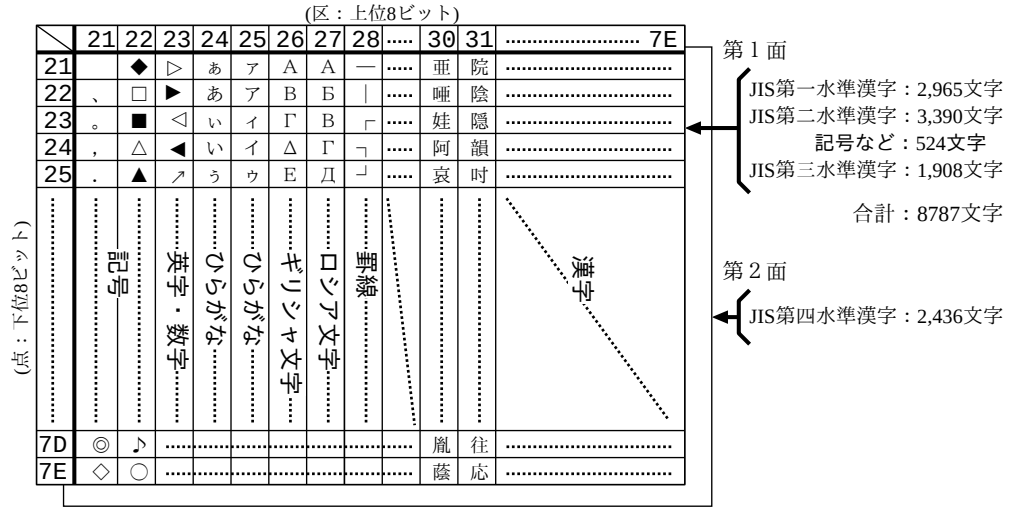


図 5: JIS 拡張漢字コード表

3. 日本で使用される文字符号化方式 (character encoding scheme)

   文字データをファイルに格納したり通信路に送信したりするとき、どんなバイト列に対応付けるか考える必要がある。

   (a) ISO-2022-JP(通称 JIS コード)

   電子メール等で用いられることが多い。エスケープシーケンスを用いて、使用する符号化文字集合 (文字コード表) を切換える (初期状態は ASCII)。7bit の範囲にエンコーディングされる。次にエスケープシーケンスの一部を示す。

ESC	(	B		:	ASCII に切換える。
ESC	(	J		:	JIS X 0201 の半角英数に切換える。
ESC	$	B		:	JIS X 0208 に切換える。
ESC	$	(	Q	:	JIS X 0213 の第１面に切換える。
ESC	$	(	P	:	JIS X 0213 の第２面に切換える。

   例：“A 亜 a\￥” を ISO-2022-JP にエンコーディングした状態

41H	1BH	24H	42H	30H	21H	1BH	28H	42H	61H	5CH	1BH	28H	4AH	5CH	1BH	28H	42H
'A'	ESC	'$'	'B'	'亜'		ESC	'('	'B'	'a'	'\'	ESC	'('	'J'	'￥'	ESC	'('	'B'
	<JIS X0208 へ>					<ASCII へ>					<JIS X0201 へ>				<ASCII へ>		

   (b) EUC-JP(Extended UNIX Code for Japanese, 日本語 EUC)

   1980 年代に UNIX 上で日本語を扱うために考案された。現在では UTF-8 が普及し使用されることが少なくなっている。エスケープシーケンスを用いることなく、ASCII と JIS X 0208 を切換える。符号化文字集合として JIS X 0201 を指定できないので、半角の'￥' を表現することができない。JIS X 0208 の文字は JIS 漢字コードに 8080H を加えた値で表現する。

   「ISO-2022-JP より短くエンコーディングできる」、「変換規則が単純」、「バイト列のどの部分が全角文字か直に分かる」、「漢字を構成するバイトが ASCII と重ならない」等のメリットがある。

   例： “A 亜 a\\” を EUC-JP にエンコーディングした状態

41H	B0H	A1H	61H	5CH	5CH
'A'	'亜'		'a'	'\'	'\'

   5CH は'\' を表現する。

   (c) Shift_JIS(Shift JIS code, SJIS)

   1980 年代に PC で日本語を扱うために考案され現在も Windows で使用されている。エスケープシーケンスを用いることなく、JIS X 0201 と JIS X 0208 を切換える。符号化文字集合として ASCII を指定できないので、半角の'\' を表現することができない[3]。JIS X 0208 の文字は、JIS 漢字コードを 8140H - 9FFCH と E040H - FCFCH に変換して表現する。計算で変換できるが若干複雑である。

   「ISO-2022-JP より短くエンコーディングできる」、「半角カナを 1 バイトで表現できる」等のメリットがある。

   例： “A 亜 a￥￥” を Shift_JIS にエンコーディングした状態

41H	88H	9FH	61H	5CH	5CH
'A'	'亜'		'a'	'￥'	'￥'

   5CH は'￥' を表現する。

[3]日本の Java 言語や C 言語の教科書で'\n' を'￥n' と表記しているのは Windows 使用前提だからか？

4. Unicode(ユニコード)

   符号化文字集合と文字符号化方式の両方を定めた文字コードの規格である。単一 (“Uni”) の巨大な文字コード表に、世界中の全ての文字を収めようとしている。符号化文字集合と文字符号化方式の両方を定めているので、符号化に関する混乱も少ない。

   当初、中国・日本・韓国の漢字を統一 (CJK 統合漢字) することで、16bit で全ての文字にコードを割り振る計画だったが、世界中の文字は予想以上に多かったため途中で 21bit に変更された。CJK 統合漢字になっているので JIS 漢字コードとの変換には変換表が必要になる。

   (a) 歴史

   1980 年代後半から検討され 1990 年代前半に実用化された。現在も世界各地の文字、古い文字 (古代文字)、新しい文字 (絵文字) 等が追加され続けている。

Unicode1.0.1	1992	16bit	28,359 文字	JIS X 0201,0208,0212 の文字収録
Unicode2.0.0	1996	21bit	38,950 文字	Unicode1.x.x との互換性を失う
Unicode3.0.0	2002	21bit	95,221 文字	JIS X 0213 へ対応
Unicode4.0.0	2003	21bit	96,447 文字	
Unicode5.0.0	2006	21bit	99,089 文字	楔文字や象形文字を追加
Unicode6.0.0	2012	21bit	109,449 文字	携帯絵文字追加
Unicode7.0.0	2014	21bit	113,021 文字	
Unicode8.0.0	2015	21bit	120,737 文字	
Unicode9.0.0	2016	21bit	128,172 文字	
Unicode10.0.0	2017	21bit	136,690 文字	変体仮名追加

   (b) 符号化文字集合

   図 6 に示すように Unicode の文字コード表は 17 面 (5bit)、256 区 (8bit)、256 点 (8bit) で構成される。この表に最大 111 万文字を収容可能である。半角 ASCII 文字から始まり、半角文字も全角文字もこの表に収録されている。

   表の 1 文字を指定するためには、21bit(5+8+8) のコード (Unicode スカラ値) を用いる。Unicode スカラ値は、`"U+1234"`のように先頭に `U+`を付加した 16 進数で表現する。

   00 面 (U+0000〜U+FFFF) は BMP(Basic Multilingul Plane, 基本多言語面) と呼ばれる。当初は BMP だけで全ての文字を格納する予定であった。U+0000〜U+007F は ASCII コードと同じ配列になっている。Java 言語の `char` 型が 16bit なのも、16bit の Unicode を格納することを前提に設計されたためである。

   なお、完全なコード一覧は、`https://ja.wikipedia.org/wiki/Unicode` で見ることができる。面白い文字が見つかるので、一度、見てほしい。

   (c) 文字符号化方式

   Unicode 用の文字符号化方式 (UTF:Unicode Transformation Format) が定められている。多くの方式が定められているが、代表的な方式は以下の 3 つである。

   UTF-8　：符号化結果は 8bit,16bit,24bit,32bit のどれかになる。
   　　最もよく使用される方式である。半角英数字が小さく符号化できる。

   UTF-16　：00 面を 16bit に、その他は 32bit に符号化する。
   　　Unicode が 21bit になってから、Java の char 型に使用されている。

   UTF-32　：Unicode の上位に 0 を付加し 32bit に符号化する。
   　　固定長なので扱いやすい。

.

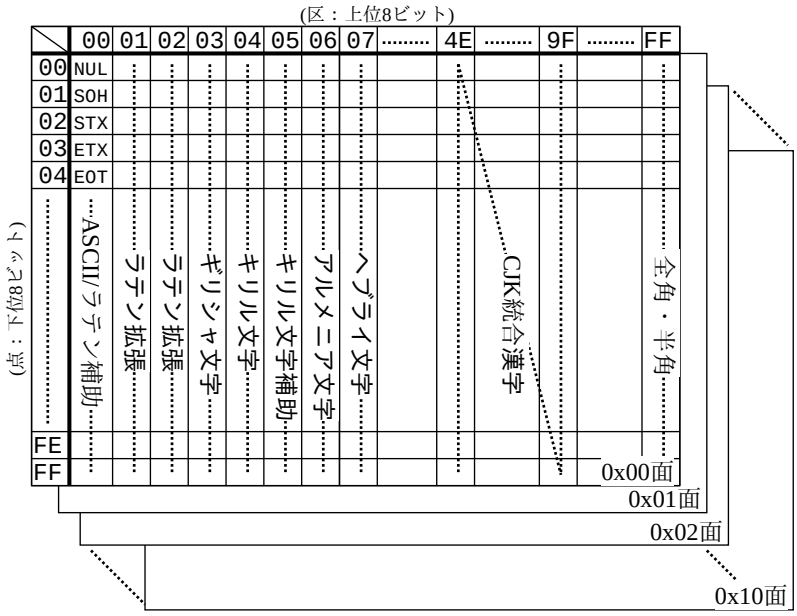


図 6: Unicode 一覧

(d) UTF-8 による符号化手順

21bit の Unicode のコードポイントを 2 進表記したものを、表中の y、x に格納する。y の部分には最低 1 つの 1 のビットがあるはずである。符号化の結果は 1〜4 バイトになる。

Unicode	UTF-8(ビット列で表記)				バイト数
U+0000〜U+007F	0xxxxxxx				1
例　U+0000	00000000	(00H)			
例　U+007F	01111111	(7FH)			
U+0080〜U+07FF	110yyyyx	10xxxxxx			2
例　U+0080	11000010	10000000	(C2H 80H)		
例　U+07FF	11011111	10111111	(DFH BFH)		
U+0800〜U+FFFF	1110yyyy	10yxxxxx	10xxxxxx		3
例　U+0800	11100000	10100000	10000000		
	(E0H)	(A0H)	(80H)		
例　U+FFFF	11101111	10111111	10111111		
	(EFH)	(BFH)	(BFH)		
U+10000〜U+1FFFFF	11110yyy	10yyxxxx	10xxxxxx	10xxxxxx	4
例　U+10000	11110000	10010000	10000000	10000000	
	(F0H)	(90H)	(80H)	(80H)	
例　U+1FFFFF	11110111	10111111	10111111	10111111	
	(F7H)	(BFH)	(BFH)	(BFH)	

5. 文字コード変換コマンド (iconv)

UNIX や Mac では、iconv[4]コマンドを使用して文字符号化方式（文字コード）を変換することができる。次に、iconv コマンドの使い方を簡単に説明する。

```
書式１：iconv [-f ENCODE] [-t ENCODE] [テキストファイル ...]

書式２：iconv -l

説明：ENCODE は文字符号化方式を表す文字列である。

        書式１の場合、-f ENCODE で指定される符号化方式のデータを入力し、
        -t ENCODE で指定される符号化方式に変換し、標準出力に出力する。
        テキストファイルが指定されない場合は標準入力からデータを入力する。

        書式２は、iconv が扱うことができる文字符号化方式の一覧を表示する。

実行例：
$ emacs utf8.txt                                          # 何かテキストファイルを作る
$ iconv -f UTF-8 -t ISO-2022-JP utf8.txt > iso.txt # ISO-2022-JP に変換する
```

6. 16 進ダンプコマンド (hexdump)

hexdump コマンドはファイルか標準入力のバイト列を 16 進数に変換して表示する。次に簡単な説明と実行例を示す。

```
書式：hexdump [オプション] [ファイル ...]

説明：hexdump は、ファイルの内容を 16 進数に変換して標準出力に出力する。
        [ファイル ...] が省略された場合はファイルの代わりに標準入力を使用する。

実行例：
$ cat ascii.txt
ABCDEFG
$ hexdump ascii.txt
0000000 41 42 43 44 45 46 47 0a
0000008
$ iconv -f UTF-8 -t UTF-32BE ascii.txt  | hexdump
0000000 00 00 00 41 00 00 00 42 00 00 00 43 00 00 00 44
0000010 00 00 00 45 00 00 00 46 00 00 00 47 00 00 00 0a
0000020
$ iconv -f UTF-8 -t UTF-32LE ascii.txt  | hexdump
0000000 41 00 00 00 42 00 00 00 43 00 00 00 44 00 00 00
0000010 45 00 00 00 46 00 00 00 47 00 00 00 0a 00 00 00
0000020
$

# UTF-32BE : ビッグエンディアン (上位桁からのバイト順)
# UTF-32LE : リトルエンディアン (下位桁からのバイト順)
```

[4]International Codeset Conversion Library に付属する文字コード変換プログラムである。

7. 宿題

半角英数字、全角カナ、全角漢字、半角バックスラッシュ(ASCII)、半角円記号 (JIS X 0201) 等、色々な文字を含むテキストファイルを観察する。

(a) ファイルの作成

IE 電算の Mac の emacs では、[￥] キーでバックスラッシュが入力される。[command]+[￥] で'\' が入力できる。

```
$ cat x0208UTF8.txt
0Aa!
あア亜院
a\￥A
$
```

(b) 様々な文字符号化方式に変換し結果を 16 進数ダンプして確認する。

```
$ iconv -f UTF-8 -t UTF-32BE x0208UTF8.txt | hexdump
0000000 00 00 00 30 00 00 00 41 00 00 00 61 00 00 00 21
0000010 00 00 00 0a 00 00 30 42 00 00 30 a2 00 00 4e 9c
0000020 00 00 96 62 00 00 00 0a 00 00 00 61 00 00 00 5c
0000030 00 00 00 a5 00 00 00 41 00 00 00 0a
000003c
$
```

(c) 16 進数ダンプの内容を解析する。

```
// UTF-32 に変換した結果(Unicode のコードポイントと対応がわかりやすい)
00 00 00 30 : 0  (U+0030) : BMP (ASCII 領域)
00 00 00 41 : A  (U+0041) : BMP (ASCII 領域)
00 00 00 61 : a  (U+0061) : BMP (ASCII 領域)
00 00 00 21 : !  (U+0021) : BMP (ASCII 領域)
00 00 00 0a : LF (U+000a) : BMP (ASCII 領域)
00 00 30 42 : あ (U+3042) : BMP (かな領域)
00 00 30 a2 : ア (U+30a2) : BMP (かな領域)
00 00 4e 9c : 亜 (U+4e9c) : BMP (CJK 統合漢字領域)
00 00 96 62 : 院(U+9662) : BMP (CJK 統合漢字領域)
00 00 00 0a : LF (U+000a) : BMP (ASCII 領域)
00 00 00 61 : a  (U+0061) : BMP (ASCII 領域)
00 00 00 5c : \  (U+005c) : BMP (ASCII 領域)
00 00 00 a5 : ￥  (U+00a5) : BMP (全角・半角領域)
00 00 00 41 : A  (U+0041) : BMP (ASCII 領域)
00 00 00 0a : LF (U+000a) : BMP (ASCII 領域)
```

(d) 上記の変換と解析を幾つかの文字符号化方式について確認する。(UTF-8、ISO-2022-JP、SJIS、 EUC-JP、...)

(e) EUC-JP に変換した後 UTF-8 に戻すとどうなるか確認する。

```
$ iconv -f UTF-8 -t EUC-JP x0208UTF8.txt | iconv -f EUC-JP -t UTF-8 | hexdump
$ iconv -f UTF-8 -t EUC-JP x0208UTF8.txt | iconv -f EUC-JP -t UTF-8
```