ユニコード (Unicode) へ

tbasic.org [*1]

[2023 年 08 月版]

コンピュータが出現してから，約 80 年が過ぎました[*2]。

現在では，コンピュータや Web 上だけでなく，多くの環境で，日本語に限らず，外国語も使うことが多くなってきました。また，Web 上でよく見かける翻訳サイトなどでは，一つのページに多国語が混在しています。

一方，Windows 上では日本語入力方式として，Shift-JIS が古くから，そして現在も良く使われています。しかし，Shift-JIS は元々日本語を表現するものとして作られています。ですから，Shift-JIS 形式のファイルでは，日本語以外，例えばフランス語の”très bien” を表すことはできません。

それでは，多国語が混在するような，文書や Web ページはどのようにして作られているのでしょうか。

実は，そのようなものの多くは，ユニコードを使って作られています。ユニコードは多国語を混在して使うことのできる仕組みです。ここではその，ユニコードについて簡単にまとめます。

[*1] https//tbasic.org

[*2] いつコンピューターが出現したかはその定義によって色々考えられます。ここでは一応，1940 年代後半として考えています。

目次

第 I 部
ユニコードへ

1　ユニコードとは

現在，私たちはコンピューターでワープロ，メール等，色々なソフトウエアを使用する場合，入力の問題を別にすれば，日本語を特に意識せずに使うことができます。しかし実は日本語に限らず，アルファベットや数字などもコンピューターで使うためには，コンピューター内部ではいくつかの仕組みが必要です。ただ日本で販売されているコンピューターは日本語を使えるような仕様で作られていますから，日本語だけを使用している場合は，そのことを特に意識する必要はありません。

しかし，いくつかの言語を同時に使う場合は，それらの仕組みを意識する必要があります。例えば，中国語やハングルなどを使った文書を取り扱う場合，それに対応した処理を考えることになります。元々は，それぞれの言語に従って，それぞれの方式があり，それぞれを切り替えながら利用することになります。しかしそのように利用するにしても，利用するソフトウエアがそれに対応していなければ，それを利用することはできません。例えば，Tiny Basic for Windows でも，Ver. 1.17 までは，

`Print` ”你好”

のようなコマンドは正しく実行できませんでした。

これは当時の Tiny Basic for Windows が日本語の漢字でない ”你” を正しく扱えないことに依りました。現在，世界は国境を越えての情報交換が頻繁に行われています。そして，これから益々複数の国々の文字を扱うことが多くなってくるでしょう。この問題への対応として，世界中の文字を同時に扱う仕組みが求められていました。これの一つの解決策がユニコードでした。ユニコードは世界中の文字を一つの仕組みの中で表す試みです。この方式を利用すれは一つのプログラムの中に色々な言語が混在して利用できるようになります。

例えば，現在の Tiny Basic for Windows では，ユニコード対応ですので

`Print` ”你好!，안녕하십니까 세계！，Hello World!，こんにちは!”

と入力して実行すると，画面には

你好!，안녕하십니까 세계 !，Hello World!，こんにちは!

と表示されます。これは Tiny Basic for Windows がユニコードに対応していることに依ります。まとめると，

- コンピューターで文字を扱うには対応する仕組みが必要
- 元々は言語ごとに，その仕組みが考えられていた。
- それらを統一して利用できる方法として考えられたのがユニコード

と言えるでしょう。この文書では日本語とユニコードについて簡単にまとめます。ユニコードについての詳細で正確な理解には，ユニコード規格の本家

Unicode Technical Site（`https://unicode.org/main.html`）

の参照を薦めます。

2　標準化への動き

1940 年代後半に開発が始まったコンピューターは，1960 年代になると，多くの国々で使われるようになりました。それらの国々ではコンピューターを使ってデータの保存や通信等が行われました。しかし，初期のコンピューターは機種ごとに出入力仕様が異なっていたため，異機種でのデータ交換は困難でした。

コンピューターの利用が広がるにつれ，それら異なるコンピュータ間でのデータの共有等の必要性が生じ，特定の装置に依存しない仕様，データ形式の標準化が模索されるようになりました。

1961 年，米国や，ヨーロッパで標準化への検討が始まりました。1962 年 ISO(国際標準化機構) は，文字符号の標準化の検討を始め，米国は ASA [*3]での検討を基にした 7bit 案を，イギリスは 6bit 案を ISO に提案しました。これを基に，ISO は 1963 年に，6bit 案と 7bit 案を検討することを決めました。これらの案は，国際規格として設定されるもので，各国の標準化法に従って，幾つかの記号について複数の多様性を許容するものでした。

これらの案の内，6bit 案は，少しの修正を経て Ecma-1 規格[*4]として，ヨーロッパで採用されました

2.1　7 ビット符号

2.1.1　ASCII

米国では ASA によって 7bit 案をもとにした規格：ASCII(American Standard Code for Information Interchange) の初版が 1963 年制定されました。

ASCII はコンピューターだけでなく，当時使われていた電信による情報交換のためのテレタイプ端末等の標準規格としても想定されていました。それまで使われていたいくつかの規格を含むように 7 ビットコードで規定し，0(00)〜127(7F) までの数値で番号付けされています[*5]。1963 年に 1 次規格，1965 年に 2 次規格，1967 年に 3 次規格（最終規格）が規定され現在に至っています[*6]。3 次規格では，制御文字として，0(00)〜31(1F)，127(7F) を制御文字，32(20)〜126(7E) を**印刷可能文字**として規定されています[*7]。

ASCII は，制御文字，英数字といくつかの記号を定めた符号化文字集合の最も基本的なものですが，現在でも現役で利用されています。

次ページの表は，ASCII 符号を表すために，7 ビットで表される数 0〜127 を 16×8 の表に表したものです。行及び列を表す数は 10 進法で表されていますが，それらの数は 7 ビットをそれぞれ，行の数字 (00〜15) は下位 4 ビット，列の数字 (00〜07) は上位 3 ビットを表したものです。ですから，対応する位置の番号は，行の数 + 列の数 ×16 で得られます。例えば，SP とある位置を示す数は，$00 + 02 \times 16 = 32$ となります。また，DEL とある位置を示す数は，$15 + 07 \times 16 = 127$ となります。

[*3] ASA: 米国規格協会: ワシントン D.C. にある標準化団体，現在では，ANSI: 米国国家規格協会

[*4] Ecma International: スイスのジュネーブにある情報通信等の標準化団体は 1961 年に設立。

[*5] () 内の数値は 16 進数表示。

[*6] 英子文字が規定されたのは 2 次規格からです。

[*7] 32(20 SP) を印刷可能文字としない考えもあります。

ASCII 符号

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

ここで，制御文字の名称は以下の通りです。

ASCII 符号（制御文字部分）

簡略名	名称	簡略名	名称
NULL	Null	DLE	Data Link Escape
SOH	Start of Heading	DC1	Device Control 1 (often XON)
STX	Start of Text	DC2	Device Control 2
ETX	End of Text	DC3	Device Control 3 (often XOFF)
EOT	End of Transmission	DC4	Device Control 4
ENQ	Enquiry	NAK	Negative Acknowledgement
ACK	Acknowledgement	SYN	Synchronous Idle
BELL	Bell	ETB	End of Transmission Block
BS	Backspace	CAN	Cancel
HT	Horizontal Tab	EM	End of Medium
LF	Line Feed	SUB	Substitute
VT	Vertical Tab	ESC	Escape
FF	Form Feed	FS	File Separator
CR	Carriage Return	GS	Group Separator
SO	Shift Out	RS	Record Separator
SI	Shift In	US	Unit Separator
DEL	Delete		

制御文字は，元々はテレタイプ端末の動作を指示するもので，上記で名称の項に表示されている意味・動作をしました。しかし，現在は名称にある操作に限らず，状況に応じて，別の意味を示すことがあります。

2.1.2 ISO/IEC 646（ECMA-6）

1963 年米国では 7 ビットの ASCII が制定されましたが，ヨーロッパでも 7 ビット案が模索されていました。1965 年 7 ビット規格 ECMA-6 が制定されました。以後，いくつかの更新版（ISO/IEC 646）が ISO との共同で制定されました[*8]。1991 年 ISO/IEC 646 1991（ECMA-6 6th edition 1991）が最終版となりました。

ISO/IEC 646 は ASCII の国際版と言えるもので，いくつかの文字（以下で①，②で示される部分）が変更可能になっています。①の部分については，変更に制限があります。また，②は各国の状況によって指定可能とされる部分で，多くの ISO/IEC 646 準拠規格があります。特に，IRV（International Reference Version）といわれる準拠規格は ASCII と同じになっています。

ISO/IEC 646

	00	01	02	03	04	05	06	07
00	NUL	DLE	SP	0	@	P	‘	p
01	SOH	DC1	!	1	A	Q	a	q
02	STX	DC2	”	2	B	R	b	r
03	ETX	DC3	①	3	C	S	c	s
04	EOT	DC4	①	4	D	T	d	t
05	ENQ	NAK	%	5	E	U	e	u
06	ACK	SYN	&	6	F	V	f	v
07	BEL	ETB	’	7	G	W	g	w
08	BS	CAN	(	8	H	X	h	x
09	HT	EM	)	9	I	Y	i	y
10	LF	SUB	*	:	J	②	j	z
11	VT	ESC	+	;	K	②	k	②
12	FF	IS_4	,	<	L	②	l	②
13	CR	IS_3	-	=	M	②	m	②
14	SO	IS_2	.	>	N	②	n	②
15	SI	IS_1	/	?	O	_	o	DEL

ASCII と ISO/IEC 646 の変更可能な部分①

番号（16 進）	番号（10 進）	ASCII	ISO/IEC 646
23	35	#	# or £
24	36	$	$ or ¤

[*8] IEC（International Electrotechnical Commission）国際電気標準会議，電気関係の国際標準を扱う標準化団体。情報系技術については ISO と共同作業も行っています。

2.1.3 JIS X 0201

ISO/IEC 646 に準拠した日本規格として，JIS X 0201 があります。JIS X 0201 の正式名称は，「7 ビット及び 8 ビットの情報交換用符号化文字集合」で 1969 年の初版制定です。この規格の 7 ビット部分が，ISO/IEC 646 に準拠した規格になっています。ASCII との違いは 2 か所です。

ASCII と JIS X 0201 の異なる部分（②の部分）

番号（16 進）	番号（10 進）	ASCII	JIS X0201
5C	92	\	¥
7E	126	~	‾

2.2 8 ビット符号

1960 年代中頃，基本処理単位が 8 ビット（=バイト[*9]）のコンピュータが主流になり始めました[*10][*11]。

7 ビットで構成された ASCII コードは，英語を扱うにはほぼ十分な文字を提供しますが，他のヨーロッパの言語に限ってみても，十分とは言えません。8 ビット処理のコンピュータが主流になるにつれて，7 ビット ASCII（或いは ISO/IEC 646）の 8 ビットへの拡張が始まりました。

その結果 1960 年代後半には多くの国々で，自国言語に適用した文字集合が使われるようになりました。そのような中で，日本でもコンピューター上でカタカナが使われ始めました[*12]。

2.2.1 JIS X 0201

日本では 1969 年，半角カタカナを ASCII に追加した[*13]ANK コードが制定されました[*14]。これは 1987 年 JIS X 0201 と改名されて，現在に至っています。

JIS X 0201 の正式名称は，「7 ビット及び 8 ビットの情報交換用符号化文字集合」で，7 ビット及び 8 ビットの文字集合を規定しています。は次の通りです。

次の表は，JIS X 0201 の 8 ビット符号化文字集合を 8 ビットで表される数 0〜255 を 16×16 の表で表したものです。行及び列を表す数は 10 進法で表されていますが，それらの数は 8 ビットをそれぞれ，行の数字は下位 4 ビット，列の数字は上位 4 ビットを表したものです。ですから，対応する位置の番号は，行の数 + 列の数 ×16 で得られます。例えば，SP を示す数は，$00 + 02 \times 16 = 32$ となります。また，DEL を示す数は，$15 + 07 \times 16 = 127$ となります。更に，「ｱ」を示す数は，$01 + 11 \times 16 = 177$ となります。

[*9] バイトは元々は，コンピュータでの基本処理単位を表すもので，8 とは限りませんでした。現在では，バイトと言うと普通 8 ビットを表します。

[*10] 1964 年に発売された IBM の System/360 は 8 ビットを 1 バイトとする 32 ビットコンピュータでした。以後，アドレス指定をバイト単位で行うものは後の標準となりました。

[*11] ここでの，コンピュータでの基本処理単位が 8 ビットとは，CPU のビット数とは別のことです。現在の CPU は 64 ビットが主流ですが，現在でもアドレス指定等はバイト単位，即ち 8 ビット単位で行われています。

[*12] 実際，IBM の System/360 では，1964 年発売当時既に EBCDIC 上でカタカナが導入されたものがありました。

[*13] 正確には JIS X 0201 に追加した

[*14] このコードは ASCII を 8 ビットに拡張した日本のコードと言う意味から，JISCII（Japanese Industrial Standard Code for Information Interchange）と呼ばれることもありました。

JIS X 0201 の 8bit 符号表の領域

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
00	NUL	DLK	SP	0	@	P	‘	p	未定義	DCS	未定義	ー	タ	ミ		
01	SOH	DC1	!	1	A	Q	a	q	未定義	PU1	。	ア	チ	ム		
02	STX	DC2	”	2	B	R	b	r	BPH	PU2	「	イ	ツ	メ		
03	ETX	DC3	#	3	C	S	c	s	NBH	STS	」	ウ	テ	モ		
04	EOT	DC4	$	4	D	T	d	t	未定義	CCH	、	エ	ト	ヤ		
05	ENQ	NAK	%	5	E	U	e	u	NEL	MW	・	オ	ナ	ユ		
06	ACK	SYN	&	6	F	V	f	v	SSA	SPA	ヲ	カ	ニ	ヨ		
07	BEL	ETB	’	7	G	W	g	w	ESA	EPA	ァ	キ	ヌ	ラ		
08	BS	CAN	(	8	H	X	h	x	HTS	SOS	ィ	ク	ネ	リ		
09	HT	EM	)	9	I	Y	i	y	HTJ	未定義	ゥ	ケ	ノ	ル		
10	LF	SUB	*	:	J	Z	j	z	VTS	SCI	ェ	コ	ハ	レ		
11	VT	ESC	+	;	K	[	k	{	PLD	CSI	ォ	サ	ヒ	ロ		
12	FF	IS4	<	<	L	¥	l	|	PLU	ST	ャ	シ	フ	ワ		
13	CR	IS3	-	=	M	]	m	}	RI	OSC	ュ	ス	ヘ	ン		
14	SO	IS2	.	>	N	^	n	¯	SS2	PM	ョ	セ	ホ	゛		
15	SI	IS1	/	?	O	_	o	DEL	SS3	APC	ッ	ソ	マ	゜		未定義

1970 年代に入ると，大型機からオフコンのような中型コンピュータ，更にパソコンのような小型コンピュータが世界中へ普及し始めました。それに伴い各コンピュータの各国への対応版が作成され，種々の言語対応への要求が高まり，その結果，種々の８ビット文字集合が定められ，利用されました。それらの文字集合間の変換は可能であっても，煩雑であり，そのため同機種間のデータのやり取りが主なものでした。

2.2.2 ECMA-94

1982 年頃 ECMA と ANSI X3L2 [*15]では，8 ビット文字集合の標準化が緊急の課題であるとの認識で，議論の交換を行いました。その議論の中で，1984 年 ECMA は一つの案を策定し，ISO に提案しました。ECMA はそれを ECMA-94 とし，1985 年に第１版を出版しました。この規格はいくつかの言語を一つの符号表で表すという考え方で作られています。

第１版では，Latin1 アルファベットとして，西ヨーロッパで広く使われている文字記号が登録されました。1986 年第２版では，Latin2，Latin3，Latin4 アルファベットとして，中央ヨーロッパ，南ヨーロッパ，北ヨーロッパの文字記号が登録されました。

[*15] X3L2 は ANSI の技術委員会の１つ。

次は Latin1 の符号表です。

ECMA-94（1995 年）Latin1 符号表

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
00			SP	0	@	P	‘	p			NBSP	°	À	Đ	à	ð
01			!	1	A	Q	a	q			¡	±	Á	Ñ	á	ñ
02			”	2	B	R	b	r			¢	²	Â	Ò	â	ò
03			#	3	C	S	c	s			£	³	Ã	Ó	ã	ó
04			$	4	D	T	d	t			¤	´	Ä	Ô	ä	ô
05			%	5	E	U	e	u			¥	µ	Å	Õ	å	õ
06			&	6	F	V	f	v			¦	¶	Æ	Ö	æ	ö
07			’	7	G	W	g	w			§	·	Ç	×	ç	÷
08			(	8	H	X	h	x			¨	¸	È	Ø	è	ø
09			)	9	I	Y	i	y			©	¹	É	Ù	é	ù
10			*	:	J	Z	j	z			ª	º	Ê	Ú	ê	ú
11			+	;	K	[	k	{			«	»	Ë	Û	ë	û
12			,	<	L	\	l	|			¬	¼	Ì	Ü	ì	ü
13			-	=	M	]	m	}			SHY	½	Í	Ý	í	ý
14			.	>	N	^	n	~			®	¾	Î	Þ	î	þ
15			/	?	O	_	o	DEL			¯	¿	Ï	ß	ï	ÿ

これらの符号表では，文字は制御文字部分以外，印刷可能部分に割り当てられています。

2.2.3 ISO/IEC 8859

ECMA は ECMA-94 を ISO/IEC に提案し，その後このプロジェクトは ISO/IEC と ECMA との共同で進められ，Latin1,2 は 1987 年 ISO/IEC 8859（1987）として制定されました。その後 ISO/IEC 8859 は，2009 年まで更新が続けられ，全部で 15 の文字集合が規定されました。それらはそれぞれ ISO/IEC 8859-n と名付けられました。例えば，ECMA-94（1995 年）Latin1 は，ISO/IEC 8859-1 と名付けられています。

ISO/IEC 8859-n の中には日本文字用は含まれていませんが，JIS X 0201 はその方針に沿った規格になっています。

2.3 文字集合の切り替え（ISO/IEC 2022）

コンピューターの利用が盛んになるにつれ，各国で色々な文字集合が使われるようになりました。しかしそれらは当初，各機種独自の仕様に基づくものが多く，異なる機種や異なる国での相互の利用は限られていました。

1960 年代後半になると，大型機やオフィスコンピューターなどの利用が各国で広がり，それら色々な文字集合を使った文書を異なるコンピューターや異なる国々で相互に利用することへの要望が高まりました。

ISO/IEC 2022 はそのような要望への ISO の対応です。

ISO/IEC 2022 は，機種や国に関わらず，文字集合を切り替えて使うための共通仕様として，扱う文字集合の型，それを符号として表す方法，種々の処理規則等を規定しています。

1973 年初版のタイトルは「ISO 7 ビット文字符号のための拡張法」[*16]で主に ISO/IEC 646 型の文字集合を対象としていましたが，8 ビットへの拡張も考察されていました。1982 年第 2 版，1986 年第 3 版のタイトルは「情報処理 — ISO 7 ビットおよび 8 ビット文字符号 — コード拡張法」[*17]，1994 年第 4 版のタイトルは「情報技術 – 文字符号の構造及び拡張法」[*18] となっています。タイトルの変遷を見るだけで，7 ビットから，7 ビット 8 ビット併用，そして 8 ビットが標準となることが分かります。

日本では，ISO/IEC 2022 の対応翻訳規格 JIS X 0202 が制定されています。

そこでは，規格の目的として，「文字集合の符号化のための 8 ビット符号および 7 ビット符号の構造を規定する。」とあります。そして，この構造技術の使用は次の利点を持つと，主張しています。

- 情報処理システム設計における符号構造に対する同一の形式での規定を可能にする。
- 合意された文字集合の呼び出しの標準化された方法を提供する。
- 8 ビット及び 7 ビット符号の文字集合を用いる環境間で相互のデータ交換を可能にする。
- 相互操作が要求されるシステム間の矛盾の危険性を軽減する。

この技術で対象となる符号表の領域は 8 ビットでは次の領域です[*19]。

[*16] Code Extension techniques for use with th ISO 7-bit coded character set
[*17] Information processing — ISO 7-bit and 8-bit coded character sets —Code extension techniques
[*18] Information technology – Character code structure and extension techniques
[*19] 7 ビットも対象としていますがここでは，8 ビットのみ説明します。

8bit 符号表の領域

	00 01	02 03 04 05 06 07	08 09	10 11 12 13 14 15
00	CL 領域	SP	CR 領域	未定義
01				
02				
03				
04				
05				
06				
07				
08		GL 領域		GR 領域
09				
10				
11				
12				
13				
14				
15		DEL		未定義

ここで，CL 領域は，0〜31 の位置の領域を表します。また，GL 領域は，33〜126 の位置の領域を表し，CR 領域，GR 領域はそれぞれ，128〜159，161〜254 の位置の領域を表します。

CL，CR 領域は文字の切り替え等に関連する制御コードを格納します。ISO/IEC 2022 では，この領域に格納する種々の制御コードを規定しています。

GL，GR 領域は文字集合の位置情報を格納します。GL 領域は SP と DEL を除いて 94 種類の 8 ビットデータが格納できます。また GR 領域は 94 種類または 96 種類の 8 ビットデータが格納できます。

GR 領域には文字情報を格納しますが，その文字情報は，符号化文字集合と言う，例えば

- 1 列に並んだ 94 個の文字の集合
- 94×94 の表で表される文字の集合

のように番号付けられた文字の表を使います。

ISO/IEC 2022 では，文字の切り替え方法を細かく規定していますが，具体的な文字の切り替えの方法については，述べていません。それは，各国，各機関が必要に応じてこれらの規則方法を使って，規定することになっています[*20]。

ISO/IEC 2022 で使われる文字の集合を符号化文字集合と言いますが，一般に 94^n または，96^n の型の文字の表になります。ISO/IEC 2022 では，具体的な符号化文字集合は規定していませんから，利用する各国，機関でそれを規定する必要があります。

基本的な符号化文字集合の例としては次が挙げられます。

[*20] 具体的な例としては JIS や EUC などがありますが，これらについては以下の項で説明します。

例 2.1.

- 最も基本的な符号化文字集合は，ASCII コード表の印刷可能文字の部分です。実際，これは 94 個の文字が 1 列に並んだ表と見られます。
- JIS X 0201 のカタカナ 161 から 223 の部分は，63 個の文字で，これは 94 個の文字が 1 列に並んだ表の一部と見られます。
- ECMA-94 の Latin1 符号表の 160〜255 の部分は 96 個で，これは 96 個の文字が 1 列に並んだ表と見られます。

3　日本語環境

ここでは，日本語の利用に限定した状況について説明します。

3.1　符号化文字集合

1970 年代後半になると，ISO/IEC 2022 の技術を使って日本語を表示する環境の整備が進みました。ここでは，日本語での符号化文字集合について説明します。

3.1.1　JIS 基本漢字（JIS X 0208）[*21]

コンピュータ上で日本語を利用のための符号化文字集合は 1978 年に JIS C 6226 として制定されました。その後，1983 年に改定，1987 年に JIS X 0208 に移行，1997 年，2012 年に改定され，これが最終版です。

JIS X 0208 は 1 バイト文字として，JIS X 0201 を含むとして制定されています。

漢字集合は 2 バイト文字として，縦横 94 の表として指定しています[*22]。行を区，列を点として表します。漢字集合には，特殊文字，数字，ラテン文字，かな等が含まれ，漢字を含む図形文字 6879 文字が登録されています。

漢字は，16 区から 47 区までの第 1 水準 2,965 文字，および 48 区から 84 区までの第 2 水準 3,390 文字の合計 6,355 文字です。

全角半角の規定は厳密には規定されていませんが，JIS X 0201 は半角，図形文字は全角とされることが多いようです。以下では，図形文字はすべて全角文字として表示します。

次は，1 区〜4 区の一覧です。

JIS X 0208 符号化文字集合 1 区 1〜94 点

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11	+12	+13	+14	+15
0			、	。	,	.	・	:	;	?	!	゛	゜	´	`	¨
16	^	￣	_	ヽ	ヾ	ゝ	ゞ	〃	仝	々	〆	〇	ー	―	‐	／
32	＼	～	∥	｜	…	‥	‘	’	“	”	（	）	〔	〕	［	］
48	｛	｝	〈	〉	《	》	「	」	『	』	【	】	＋	－	±	×
64	÷	＝	≠	＜	＞	≦	≧	∞	∴	♂	♀	°	′	″	℃	¥
80	$	¢	£	%	#	&	*	@	§	☆	★	○	●	◎	◇	

これによれば，全角の「＋」は 1 区 60 点（或いは 1-60）と表されます。区点をコンピュータ上で表現する場合は，区を 1 バイト目，点を 2 バイト目で表すことになっていますが，具体的に，どのような数値で表すかは，変換方式に依存します。

[*21] JIS 漢字コード，JIS 漢字，JIS 第 1 第 2 水準漢字と呼ばれることもあります。

[*22] ISO/IEC 2022 の 94×94 の表です。

JIS X 0208 符号化文字集合 2 区 1〜94 点

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11	+12	+13	+14	+15
0		◆	□	■	△	▲	▽	▼	※	〒	→	←	↑	↓	〓	
16											∈	∋	⊆	⊇	⊂	⊃
32	∪	∩									∧	∨	¬	⇒	⇔	∀
48	∃												∠	⊥	⌒	∂
64	∇	≡	≒	≪	≫	√	∽	∝	∵	∫	∬					
80			Å	‰	♯	♭	♪	†	‡	¶					◯	

JIS X 0208 符号化文字集合 3 区 1〜94 点

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11	+12	+13	+14	+15
0																
16	0	1	2	3	4	5	6	7	8	9						
32		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
48	P	Q	R	S	T	U	V	W	X	Y	Z					
64		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
80	p	q	r	s	t	u	v	w	x	y	z					

JIS X 0208 符号化文字集合 4 区 1〜94 点

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11	+12	+13	+14	+15
0		ぁ	あ	ぃ	い	ぅ	う	ぇ	え	ぉ	お	か	が	き	ぎ	く
16	ぐ	け	げ	こ	ご	さ	ざ	し	じ	す	ず	せ	ぜ	そ	ぞ	た
32	だ	ち	ぢ	っ	つ	づ	て	で	と	ど	な	に	ぬ	ね	の	は
48	ば	ぱ	ひ	び	ぴ	ふ	ぶ	ぷ	へ	べ	ぺ	ほ	ぼ	ぽ	ま	み
64	む	め	も	ゃ	や	ゅ	ゆ	ょ	よ	ら	り	る	れ	ろ	ゎ	わ
80	ゐ	ゑ	を	ん												

16 区から 47 区 51 点までは，第 1 水準の漢字が収められています。

JIS X 0208 符号化文字集合 16 区 1〜94 点

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11	+12	+13	+14	+15
0		亜	唖	娃	阿	哀	愛	挨	姶	逢	葵	茜	穐	悪	握	渥
16	旭	葦	芦	鯵	梓	圧	斡	扱	宛	姐	虻	飴	絢	綾	鮎	或
32	粟	袷	安	庵	按	暗	案	闇	鞍	杏	以	伊	位	依	偉	囲
48	夷	委	威	尉	惟	意	慰	易	椅	為	畏	異	移	維	緯	胃
64	萎	衣	謂	違	遺	医	井	亥	域	育	郁	磯	一	壱	溢	逸
80	稲	茨	芋	鰯	允	印	咽	員	因	姻	引	飲	淫	胤	蔭	

48 区から 84 区 6 点までは，第 2 水準の漢字が収められています。

JIS X 0208 符号化文字集合 48 区 1〜94 点

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11	+12	+13	+14	+15
0		弋	丐	丕	个	丱	丶	丼	丿	乂	乖	乗	亂	亅	豫	事
16	舒	弍	于	亞	亟	亠	亢	亰	亳	亶	从	仍	仄	仆	仂	仗
32	仞	仭	仟	价	伉	佚	估	佛	佝	佗	佇	佶	侈	侏	侘	佻
48	佩	佰	侑	佯	來	侖	儘	俔	俟	俎	俘	俛	俑	俚	俐	俤
64	俥	倚	倨	倔	倪	倥	倅	伜	俶	倡	倩	倬	俾	俯	們	倆
80	偃	假	會	偕	修	偈	做	俸	倏	偸	傀	傚	傅	傴	傲	

JIS X 0208 は，日常で使うほぼ十分な漢字を含んでいます。，1980 年代のコンピューターは，JIS X 0208 が実装され，多くのコンピューターで日本語が使えるようになりました。

1980 年代から 2000 年初頭までのコンピューターでの漢字の利用は，JIS X 0208 によるものと言えます。

3.1.2 JIS X 0212（補助漢字）

JIS X 0212 は，JIS X 0208 に無い漢字等を補ったもので，1990 年に規定されました。正式名称は「情報交換用漢字符号―補助漢字」です。JIS X 0208 に無い文字を登録しています。JIS X 0208 の補助として使われることから，この名称があります。特殊文字 21 文字，アルファベット 245 文字，漢字 5801 文字が含まれます。

次は，2 区 1〜94 点ですが，2 区で実際に収録されているのは特殊文字 21 文字です。

JIS X 0212 符号化文字集合 2 区 1〜94 点

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11	+12	+13	+14	+15
0																˘
16	ˇ	˛	˙	˝	¯	˚	˚	˜	´	˛						
32			¡	¦	¿											
48																
64												º	ª	©	®	™
80	¤	№														

6 区はダイアクリティカルマーク付きギリシア文字，7 区はキリル系アルファベット，9 区はラテン系アルファベット，10 区はダイアクリティカルマーク付きラテンアルファベットが収録されています。

16 区からが，漢字です。次は，16 区 1〜94 点です

JIS X 0212 符号化文字集合 2 区 1〜94 点

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11	+12	+13	+14	+15
0		丐	丩	丅	丌	丒	丟	丣	两	丨	丫	丱	丳	丰	丵	乀
16	乁	乂	毛	乑	乚	乜	乩	乭	乮	乲	乳	乾	亂	亍	亖	亗
32	亟	亩	亹	仃	仐	仚	仛	仠	仡	仢	仨	仯	仱	仳	仵	份
48	仾	仿	伀	伂	伃	伈	伋	伌	伒	伕	伝	众	伙	伜	伝	伞
64	伟	伡	伝	伢	伣	伤	伥	伦	伧	伩	伬	伭	伮	伱	伲	伴
80	伶	伾	佀	佅	佈	佉	佋	佌	佒	佔	佖	佗	佘	佝	佟	

上に見るように，JIS X 0212 は特殊な使用を目的としているもので，一部の特殊記号以外，日常的には，余り目にしません。また，現在普通に使われる文字変換方式で補助漢字が使えるのは EUC-JP だけですが，JIS X 0212 はオプションになっていて，実際にサポートされているとは限りません。

3.1.3 JIS X 0213（拡張漢字）

JIS X 0213 は JIS X 0208（JIS 基本漢字）を拡張した漢字集合で，2000 年に初版が制定されました。正式名称は「7 ビット及び 8 ビットの 2 バイト情報交換用符号化拡張漢字集合」で，2 面 94 区 94 点（11223 文字）で構成されています。

第 1 面は 94 区 94 点で構成され，JIS X 0208 の上位互換として配置しています[*23]。第 2 面は第 1 面と同じく 94 区 94 点で構成されていますが，そのうち文字の存在する符号領域は 1 区 (1), 3 – 5 区 (3), 8 区 (1), 12 – 15 区 (4), 78 – 94 区 (17) に限られ，合計 26 点区です。

漢字では，第 1 面に JIS X 0208 で空いている部分に追加した漢字を第 3 次水準漢字と定めています。第 2 面の漢字を第 4 次水準漢字と定めています。2004 年，2012 年に改正され，現在に至っています。

第 1 面 14 区，15 区，47 区 52〜94 点，84 区 7 点〜94 点，85 区〜94 区が第 3 水準漢字です。

次は，1 面 1 区，1 面 4 区の一覧です。

JIS X 0213 符号化文字集合 1 面 1 区 1〜94 点

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11	+12	+13	+14	+15
0			、	。	,	.	・	:	;	?	!	゛	゜	´	`	¨
16	^	￣	_	ヽ	ヾ	ゝ	ゞ	〃	仝	々	〆	〇	ー	―	‐	/
32	\	～	∥	|	…	‥	‘	’	“	”	（	）	〔	〕	[	]
48	{	}	〈	〉	《	》	「	」	『	』	【	】	+	−	±	×
64	÷	=	≠	<	>	≦	≧	∞	∴	♂	♀	°	′	″	℃	¥
80	$	¢	£	%	#	&	*	@	§	☆	★	○	●	◎	◇	

1 面 1 区は，JIS X 0208 1 区と同じです。

[*23] JIS X 0208 で登録された区点の文字は，JIS X 0213 では，第 1 面の同じ区点で登録されています

JIS X 0213 符号化文字集合 1 面 4 区 1〜91 点

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11	+12	+13	+14	+15
0		ぁ	あ	ぃ	い	ぅ	う	ぇ	え	ぉ	お	か	が	き	ぎ	く
16	ぐ	け	げ	こ	ご	さ	ざ	し	じ	す	ず	せ	ぜ	そ	ぞ	た
32	だ	ち	ぢ	っ	つ	づ	て	で	と	ど	な	に	ぬ	ね	の	は
48	ば	ぱ	ひ	び	ぴ	ふ	ぶ	ぷ	へ	べ	ぺ	ほ	ぼ	ぽ	ま	み
64	む	め	も	ゃ	や	ゅ	ゆ	ょ	よ	ら	り	る	れ	ろ	ゎ	わ
80	ゐ	ゑ	を	ん	ゔ	ゕ	ゖ	゙	゚	゛	゜	ゝ	ゞ	ー		

1 面 4 区は，JIS X 0208 4 区とほぼ同じですが，84 点〜91 点が追加されています。

1 面 14 区から第 3 水準漢字が収められています。次は，1 面 14 区です。

JIS X 0213 符号化文字集合 1 面 14 区 1〜94 点

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11	+12	+13	+14	+15
0		倶	丈	乹	亅	丰	丰	亍	仡	份	仿	伢	伋	你	佈	佉
16	佖	佟	佪	侅	侑	侻	侔	侗	侮	俉	俠	俎	併	俔	倘	倅
32	倮	倹	倻	俛	傔	傲	傻	僖	僥	僧	儆	儃	儋	儞	儻	兗
48	免	兕	兗	貝	冝	凃	凊	凞	凡	凩	刁	刋	刋	刧	剉	剝
64	剡	劓	勫	勉	勧	勣	勛	勝	勰	匀	匀	匆	匝	卑	卑	卡
80	卣	卽	厓	厝	厲	吒	吧	呿	咤	咕	呴	呿	咻	咖	哢	

2 面は第 4 水準漢字が収められています。次は，2 面 1 区です。

JIS X 0213 符号化文字集合 2 面 1 区 1〜94 点

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11	+12	+13	+14	+15
0		𠂉	丂	丐	刋	刂	丫	丮	乀	毛	么	乑	承	月	臼	乚
16	乩	叟	充	襄	亹	亻	𠆢	亼	仃	仍	仐	仏	仚	伶	仵	伀
32	伂	伍	伷	伾	佔	佘	侀	侭	佸	侁	佽	侘	侅	俟	俑	俦
48	俜	俸	俅	俚	俏	俒	俙	俯	俥	倏	倓	倜	倰	倢	倮	倫
64	倳	偎	偟	偺	倍	偑	偪	偰	偠	偰	偢	偝	傕	傖	傜	傺
80	僤	傱	際	傻	僄	僇	僳	僿	僎	僕	僔	僃	僡	僩	僚	

2000 年代になると JIS X 0213 に多くの OS で対応するようになり，現在のコンピューターでは，JIS X 0213:2004 が利用できるようになっています。

3.2 日本語の文字符号化方式

3.2.1 テキストファイル

テキストファイルは，単純に文字を並べた集まりを保存するもの，即ち，書式なし文字列を保存する順編成ファイル[*24]と言えます。そこに含まれるのは，文字の情報，改行やタブ等の制御情報等です。これは現在のコンピューターでの扱いでは，1 列に並んだバイトの集まりです。バイト列は単に 0〜255 の数値ですから，この数値を文字として解釈しなければなりません。この解釈法をエンコーディング (Encoding) と言います。格納された文字がアルファベットだけであれば，例えば ACII コードとして読めば文字列として解釈することができます。しかし，漢字等や種々の記号を含む場合は，そう単純ではありません。使用する漢字の集合（符号化文字集合）や，それを使った解釈法の合意が必要になります。もちろん日本語以外でも状況は同じです。

即ち，テキストファイルとは，

・テキストファイル：
　書式なし文字列を保存する順編成ファイル。
・漢字や種々の国の文字を含むものもテキストファイルとして扱える。
・テキストファイルを読み書きするには対応するエンコーディングが必要。

となります。

3.2.2 ASCII

ASCII は日本語の文字入力方式ではありませんが，すべての入力方式の原点です。ASCII は元々，デジタル通信の送受信端末，テレタイププリンタの仕様書として規定され，ASCII の符号化文字集合は符号化入力方式と一体です。ですから，入力方式は，ASCII のコード番号をそのまま 7 ビット或いは 8 ビット（1 バイト）として利用します。

例 3.1.

例えば，「ASCII Code」をバイトコードで表してみます。ASCII コード番号表を参照すれば，

A	S	C	I	I		C	o	d	e
41	53	43	49	49	20	43	6F	64	65

となります。

[*24] 順編成ファイルは読み書きをする場合，先頭から後方に向かって順次 1 列に処理するファイルです。

3.2.3 JIS(ISO-2022-JP)

電子メール等で利用されている 7 ビット符号化方式です。1980 年代中頃 JUNET [*25]で開発された JUNET コードで，ニュースグループで使用されていました。1993 年 RFC1648 で規定され，1996 年，JIS X208 でも規定されています[*26]。

ISO-2022-JP の名称の通り，ISO/IEC 2022 の 7 ビット技術を利用しています。CL 領域に制御コードを配置し，GL 領域に文字集合の位置情報を配置しています。

ASCII 文字，JISX 0201 のラテン文字，JIS X 0208 が使用できます。

ASCII をスタートとし，ESC ＋ 2 バイト記号を利用して，次のように，JIS X 0201 のラテン文字，JIS X 0208 を切り替えます。

記号表記	16 進表記	意味
ESC (B	1B 28 42	ASCII
ESC (J	1B 28 4A	JIS X 0201-1976 (”Roman” set)
ESC $ @	1B 24 40	JIS X 0208-1978
ESC $ B	1B 24 42	JIS X 0208-1983

実際の文字の指定は，１バイト，２バイトいずれの場合も GL 領域に文字番号を指定します。

具体的には，ASCII と JIS X 0201 はそこでのコード番号を指定します。また，JIS X 0208 については，区番号、点番号にそれぞれ 20(16 進数) を加えたものを指定します。ここで，区点番号は 94 以下ですからこれらはすべて，10 進で 126 以下になります。つまり，GL 領域の番号になります。ですから，構成されるバイト列は先頭ビットが 0，即ち 7 ビット列になります。

上の構成法から，ISO-2022-JP では，JIS X 0201 の片仮名文字集合は利用できません。即ち，ISO-2022-JP のエンコーディングを使ったメールでは半角カタカナが使えません。

例 3.2. 例えば，「1+1 の計算」を JIS コードで表してみます。

- 「1+1」の部分は ASCII ですから，ASCII の番号から，31 2B 31 です。
- 「の計算」の部分は JIS X 0208 ですから，最新の JIS X 0208-1983 を使って 1B 24 42 で漢字への入れ替えを指示し，「の：4 区 46 点」，「計：23 区 55 点」，「算：27 区 27 点」の区点に 20(16 進) を加えて，24 4E 37 57 3B 3B を得ます。
- 最後に ASCII に戻すために，1B 28 42 を加えます。

以上から，

31 2B 31 1B 24 42 24 4E 37 57 3B 3B 1B 28 42

が得られます。

[*25] 日本の学術用研究ネットワーク：1984 年〜1991 年

[*26] 昔からの方法ですが，電子メールが 7 ビットファイルとして規定されているため，この方式が扱いやすく，現在でも多くのメールがこの方式を利用しています。

3.2.4 シフト JIS(Shift_JIS)

シフト JIS は 1980 年代前半に策定され，MSDOS，CP/M-86 に最初に実装され，主に PC 互換機での日本語符号化方式として利用されてきました。JIS 規格としては，JIS X 0208 の 1997 年の改定での附属書で規定されています[*27]。

また，シフト JIS は，CR 領域に文字データを格納するため，ISO/IEC 2022 の規約には従っていません。

ここでは，現在でもよく利用されている，シフト JIS エンコーディングを JIS X 0208 の附属書の記述に沿って説明します。

拡張アスキー（JIS X 0201）を 1 バイトで表し，拡張アスキーで未使用の部分 (81〜9F, E0〜EF) 47 (=94/2) 個を 2 バイトコードの第 1 文字とし，40〜7E，80〜FC の部分 188(=94*2) 個のバイトを第 2 文字として，94*94 の JIS X 0208 の漢字面を表します。

1 バイト符号と 2 バイト符号の第 1 バイトの領域は次の領域です。

シフト JIS：1 バイト符号と 2 バイト符号の第 1 バイトの領域

	0	1	2	3	4	5	6	7	8 9	A	B	C	D	E	F
0	NUL	DLK	SP	0	@	P	‘	p	保留域	保留域	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	2	
3	ETX	DC3	#	3	C	S	c	s		」	ウ	テ	モ	バ	保
4	EOT	DC4	$	4	D	T	d	t		、	エ	ト	ヤ	イ	
5	ENQ	NAK	%	5	E	U	e	u		・	オ	ナ	ユ	ト	
6	ACK	SYN	&	6	F	V	f	v		ヲ	カ	ニ	ヨ	符	留
7	BEL	ETB	’	7	G	W	g	w		ァ	キ	ヌ	ラ	号	
8	BS	CAN	(	8	H	X	h	x		ィ	ク	ネ	リ	の	
9	HT	EM	)	9	I	Y	i	y		ゥ	ケ	ノ	ル	第	域
A	LF	SUB	*	:	J	Z	j	z		ェ	コ	ハ	レ	1	
B	VT	ESC	+	;	K	[	k	{		ォ	サ	ヒ	ロ	バ	
C	FF	IS4	<	<	L	¥	l	|		ャ	シ	フ	ワ	イ	
D	CR	IS3	-	=	M	]	m	}		ュ	ス	ヘ	ン	ト	
E	SO	IS2	.	>	N	^	n	‾		ョ	セ	ホ	゛		
F	SI	IS1	/	?	O	_	o	DEL		ッ	ソ	マ	゜		

[*27] JIS X 0213 用としては，JIS X 0213 の附属書 1 で Shift_JISX0213 として規定されています。現在，シフト JIS と言った場合，JIS X 0213 用ではなくて，JIS X 0208 用のものです。

また，２バイト符号の第２バイトの領域は次の通りです。

シフト JIS：２バイト符号の第２バイトの領域

	0 1 2 3	4 5 6 7	8 9 A B C D E F
0 1 2 3 4 5 6 7 8 9 A B C D E F	保 留 域	２ バ イ ト 符 号 の 第 ２ バ イ ト 保留域	２ バ イ ト 符 号 の 第 ２ バ イ ト 保留域

２バイト符号の領域はいくつかの部分に分かれていることと，第１バイトの領域が 94/2=47，第２バイトの領域が 188=94*27 であることから，94 × 94 の表を表現するために，場合分けが必要となり，幾分複雑です。

具体的には次の規則でコード化します。

- １バイトコード（JIS X 0201）

  １バイトコードは JIS X 0201 の番号付けに従って，そのままコード化します。

コード番号 (10 進)	コード番号 (16 進)	内容
0〜31，127	00〜1F，7F	制御文字
32〜126	20〜7E	英数記号文字
161〜223	A1〜DF	半角カナ

- 2バイトコード（JIS X 0208）
  2バイトコードは JIS X 0208 の 94×94 の表を2バイトを使ってコード化します。

	コード番号 (10 進)	コード番号 (16 進)	個数
1 バイト目	129〜159，224〜239	81〜9F，E0〜EF	31+16=47 個
2 バイト目	64〜126，128〜252	40〜7E，80〜FC	63+125=188 個

  1バイト目は 47 (=94/2) 個あり，2バイト目は 188 (=94*2) 個あります。このことから，1つの1バイト目に対して，2つの2バイト目を使い2つの区を対応させることで，94×94 の表と対応付けます。
  具体的には各バイトの区点番号との対応は次のように定められています。区番号を k，点番号を t，1バイト目を S_1，2バイト目を S_2 とすると S_1，S_2 は次で計算されます。
  - S_1 の計算

k(10 進)	k(16 進)	S_1(10 進)	S_1(16 進)
1〜62	1〜3E	$(k-1)$ div $2+129$	$(k-1)$ div $2+81$
63〜94	3F〜5E	$(k-1)$ div $2+193$	$(k-1)$ div $2+C1$

    ここで，div 2 は 2 で割ったときの商を表します。

  - S_2 の計算
    (1) k が奇数の場合

t(10 進)	t(16 進)	S_2(10 進)	S_2(16 進)
1〜63	1〜3F	$t+63$	$t+3F$
64〜94	40〜5F	$t+64$	$t+40$

    (2) k が偶数の場合

t(10 進)	t(16 進)	S_2(10 進)	S_2(16 進)
1〜94	1〜5F	$t+158$	$t+9E$

シフト JIS の計算は多少煩雑ですが，ESC を使わない方法で，比較的小さいコードになります。

例 3.3. 例えば，「1+1 の計算」をシフト JIS コードで表してみます。

- 「1+1」の部分は ASCII ですから，ASCII の番号から，31 2B 31 です。
- 「の計算」の部分は JIS X 0208 ですから，「の：4 区 46 点」，「計：23 区 55 点」，「算：27 区 27 点」の区点に 20(16 進) に対して，それぞれ，S_1, S_2 を計算します。
  (1)「の：4 区 46 点」$S_1 = 3$ div $2+81 = 82$, $S_2 = 2E+9E = CC$(16 進)
  (2)「計：23 区 55 点」$S_1 = 22$ div $2+81 = 8C$, $S_2 = 37+3F = 76$(16 進)
  (3)「算：27 区 27 点」$S_1 = 26$ div $2+81 = 8E$, $S_2 = 1B+3F = 5A$(16 進)

以上から，

31 2B 31 82 CC 8C 76 8E 5A

が得られます。

3.2.5 シフト JIS(Shift_JISX0213)

ここでは，JIS X 0213 用として，JIS X 0213 の附属書１でとして規定されている Shift_JISX0213 について説明します[*28]。Shift_JISX0213 は Shift_JIS の拡張として定義されていますから，以下の説明では Shift_JIS と重複するところも少なくありません。

実装水準４：1 バイト符号と２バイト符号の第１バイトの領域

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLK	SP	0	@	P	‘	p	保留域		保留域	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	IS4	<	<	L	¥	l	|			ャ	シ	フ	ワ		
D	CR	IS3	-	=	M	]	m	}			ュ	ス	ヘ	ン		
E	SO	IS2	.	>	N	^	n	¯			ョ	セ	ホ	゛		保留域
F	SI	IS1	/	?	O	_	o	DEL			ッ	ソ	マ	゜		

２バイト符号の第１バイト

２バイト符号の第１バイト

[*28] このエンコーディングは，Shift_JIS2004 とも言われています。このエンコーディングが実装されているソフトウエアは現在でも余り多くはありません。

実装水準４：２バイト符号の第２バイトの領域

	0 1 2 3	4 5 6 7	8 9 A B C D E F
0 1 2 3 4 5 6 7 8 9 A B C D E F	保 留 域	２ バ イ ト 符 号 の 第 ２ バ イ ト 保留域	２ バ イ ト 符 号 の 第 ２ バ イ ト 保留域

拡張アスキー（JIS X 0201）文字を 1 バイト符号で表し，拡張アスキーで未使用の部分を 2 バイト符号の第１バイトとし，188(=94*2) 個のバイトを第２バイトとして，の JIS X 0213 の漢字面を表します。

第２面は第１面と同じように 94 区 94 点で構成されていますが，そのうち文字の存在する符号領域は 1, 3 – 5, 8, 12 – 15, 78 – 94 区，合計 26 点区に限られています。第２面を表すときはこのことを利用します。

以下，第２面を含む規格である実装水準４について説明します。

これは具体的には次の規則でコード化します。

- １バイトコード（JIS X 0201）
  １バイトコードは JIS X 0201 の番号付けに従って，そのままコード化します。

コード番号 (10 進)	コード番号 (16 進)	内容
0〜31，127	00〜1F，7F	制御文字
32〜126	20〜7E	英数記号文字
161〜223	A1〜DF	半角カナ

- 2バイトコード（JIS X 0213）

  2バイトコードは JIS X 0213 の $2 \cdot 94 \times 94$ の表を2バイトを使ってコード化します。

	コード番号 (10 進)	コード番号 (16 進)	個数
1バイト目（第1面用）	129〜159，224〜239	81〜9F，E0〜EF	31+16=47 個
1バイト目（第2面用）	240〜252	F0〜FC	13 個
2バイト目（1，2面用）	64〜126，128〜252	40〜7E，80〜FC	63+125=188 個

  1バイト目は 60 個ありますが，そのうち，47（=94/2）個を第1面の第1バイトとして使います。残りの 13 個 (F0〜FC) を第2面の第1バイトとして使います。2バイト目は 188（=94*2）個あります。これらを第1面，第2面用の第2バイトとして使います。

  このことから，1つの1バイト目に対して，2つの2バイト目を使い各面2つの区を対応させることで，94×94 の表と 26×94 の表を対応付けます。

  具体的には各バイトの区点番号との対応は次のように定められています。面番号を m，区番号を k，点番号を t とし，符号化表現の第1バイトを S_1，第2バイト目を S_2 とすると S_1，S_2 は次で計算されます。

  (1) S_1 の計算

m	k(10 進)	k(16 進)	S_1(16 進)
1	1〜62	1〜3E	$(k + 101)$ div 2
1	63〜94	3F〜5E	$(k + 181)$ div 2
2	1,3〜5,8,12〜15	1,3〜5,8,C〜F	$(k + 1DF)$ div $2 - (k$ div $8) \times 3$
2	78〜94	4E〜5E	$(k + 19B)$ div 2

  ここで，div 2 は 2 で割ったときの商を表します。

  (2) S_2 の計算

  (a) k が奇数の場合

t(10 進)	t(16 進)	S_2(16 進)
1〜63	1〜3F	$t + 3F$
64〜94	40〜5E	$t + 40$

  (b) k が偶数の場合

t(10 進)	t(16 進)	S_2(16 進)
1〜94	1〜5E	$t + 9E$

例 3.4. ここでは，2面 94 区 1 点の「亾」を Shift_JISX0213 で表してみます。

- $m = 2$, $k = 94$, $t = 1$ に対して，S_1, S_2 を計算します。

$$S_1 = (5E + 19B) \text{ div } 2 = 1F9 \text{ div } 2 = FC,\ \ S_2 = 1 + 9E = 9F(16進)$$

  以上から，

FC 9F

  が得られます。

3.2.6 EUC(EUC-JP)

EUC-JP は 1980 年代 Unix での漢字使用を目的として規定されたもので，ASCII，基本漢字（JIS X 208），半角カタカナ，補助漢字を利用できます[*29]。

文字集合	バイト数	制御文字	1 バイト目	2 バイト目	3 バイト目
ASCII	1	無し	20〜7E		
基本漢字	2	無し	区番号 +A0(A1〜FE)	点番号 +A0(A1〜FE)	
半角カタカナ	2	SS2	8E	A1〜DF	
補助漢字	3	SS3	8F	区番号 +A0(A1〜FF)	点番号 +A0(A1〜FF)

ASCII は 1 バイト，漢字は A1 から FE に[*30] 2 バイトで表現したもので，ISO/IEC 02022 規格に沿っています。

また，EUC は基本漢字部分に限れば，JIS エンコーディングを 8 ビット部分に移動した形になっています。ですから，ASCII と基本漢字に限れば，JIS と同様な文字が表現され，しかも制御文字を使わないエンコーディングで簡単な形をしています。

しかし，半角カタカナを使う場合は，制御コードと 1 バイトで 2 バイト，補助漢字の場合は，制御コードと漢字コード 2 バイトで合計 3 バイト必要になります。,

例 3.5. 「1+1 の計算」を EUC コードで表してみます。

- 「1+1」の部分は ASCII ですから，ASCII の番号から，31 2B 31 です。
- 「の計算」の部分は JIS X 0208 ですから，「の：4 区 46 点」,「計：23 区 55 点」,「算：27 区 27 点」の区点部分にそれぞれ，A0(16 **進**) を加えて，A4 CE B7 D7 BB BB を得ます。

以上から，

```
31 2B 31 A4 CE B7 D7 BB BB
```

が得られます。

[*29] 但し，補助漢字はオプションとする立場もあり，この部分を実装しているシステムは多くありません。
[*30] GR 領域

3.2.7 まとめ

符号化文字集合に対して，実際のコンピューターでそれらの文字を表示する方法は一通りではありません。日本語の場合，JIS 符号化，シフト JIS 符号化，EUC 符号化の 3 方法があります。JIS X 0208 については，すべての符号化で使用可能ですが，補助漢字，JIS X 0213 については，現在使用されているコンピューターでも実装されているとは限りません。

また，符号化方式は，コンピューターでの具体的な取扱い方法ですので，コンピューター内部のみでなく，ファイル形式も規定します。ですから，例えば，同じ文字でも，JIS 符号化で表現されたファイルと，シフト JIS 符号化で表現されたファイルは，実際にフィルに記録される内容が異なります。ですから，日本語を正しく取り扱うためには，それぞれその符号化法に対応した処理ができなければなりません。そうでないと正しく読み取ることや，処理することもできません。文字化けの現象は符号化方式の不対応から起きます。

まとめると，日本語の文字符号化方式は

(1) JIS(ISO-2022-JP)，シフト JIS(Shift_JIS)，EUC(EUC-JP) の 3 種類
(2) それぞれに応じた処理をしないと，文字化けが起こる。

となります。

4　ユニコードへ

4.1　始まり

符号化文字集合の拡張に伴って，その利用のためのエンコーディングが規定されても，そのエンコーディングが実際のコンピューターで広く使われるようになるとは限りません。

実際のコンピューターでの使用は，コンピューター内部でのエンコーディングの使用を伴います。そして，コンピューター内部でのエンコーディングの更新・変更は，OS 内部でのテキストの取り扱い処理の更新・変更が必要になります。

JIS X 208，JIS X 0212，JIS X 0213 はいずれも 94×94 の表として定義されています。そして，これらの規定は，ISO/IEC 2022 の技術の利用を想定しています。これは複数の表を切り替えてそれぞれの表の文字を使用する方法です。多くの表を用意すれば，それを切り替えることで，多くの文字を使用することができます。しかし，それには煩雑な切り替え処理が必要です。更に，これらのエンコーディングは，１文字が１バイト或いは２バイト，また，場合によっては３バイト必要となるなど煩雑な処理が必要でした。

このような状況は日本に限らず，世界的な問題でした。一方で，世界のグローバル化に伴い，多国語を同時に利用したいとの要望はますます高まっていきます。しかし，この方法では，使用文字を増やすたびに，OS 内部のエンコーディングを修正していく必要があり，そのたびに煩雑さが増し，速度の低下が起こり，困難が伴います。

このことへの対応として，小規模の符号化文字集合を複数個使って，多くの文字を表すのではなく，「**世界中の文字を含む大きな符号化文字集合を一つ定めて，それを使ってコンピューターで多国語の文字を扱う**」方法が検討されるようになりました。

1980 年代後半，このような目的へのプロジェクトが始まり，このコードは，Universal，uniform，unique な code の意味から Unicode（ユニコード）と言われるようになりました。

1988 年には「Unicode Standard への提案（Unicode 88)」が公開されました[*31]。

そこでは，

> Unicode は ASCII コードの拡張であり，次の性質を持つ。
> ・Unicode は固定 16 ビットコードである。
> ・世界中の文字と固定幅で１対１に対応する。

と主張しています。

4.2　Unicode Ver.1(1991)

1991 年には，多くの組織，企業が参加して Unicode 制定のための非営利国際的組織 Unicode Consortium が発足し，1991 年 10 月にはその仕様書”The Unicode Standard Version 1.0” が発表されました[*32]。これは，1988 年の提案に沿って，それを具体化したものでした。

[*31] https://unicode.org/history/unicode88.pdf
[*32] http://www.unicode.org/versions/Unicode1.0.0/

そこでは，

- Unicode 番号は，先頭に U+ を付けた 16 ビットの固定長 16 進数 U+0000 から U+FFFF で表される。
- Unicode は，現在，過去の世界中の文書を表す十分な文字を含んでいる。
- Unicode では，どのような言語であってもエスケープまたは制御記号による切り替えなしで利用できる
- 固定幅のコードは，種々のテキスト処理がより簡単に可能である。

と述べられています。

Ver. 1.0 では，全体で 34,338 の文字等[*33]が登録され，日本語では，JIS X 208（基本漢字）, JIS X 0212（補助漢字）が含まれました。

また，内部コードとして 16 ビットを利用する際，Endian [*34]を区別するものとして BOM(Byte Order Mark) が定義されました。

4.3 Unicode Ver.2(1996)

1996 年には Ver. 2.0 が発表されました[*35]。Ver. 1 の発表後，各国より多くの文字の登録の要請があり，Ver. 2 では多くの文字が追加され，合計 47,400 文字が登録されています。

Unicode は 16 ビットで表すとされていますが，この範囲では最大 $2^{16} = 65536$ までしか表すことができません。Ver.1 当時の想定ではこれでも十分とされていましたが[*36]，Ver. 2 でこれを超える文字を登録するためのサロゲートペア（Surrogate pair）と言われる拡大法を定めました。

これは，1024×1024 の表を新たに登録可能にするもので，2 個の 16 ビット (=32 ビット）を使って表現されます。この構成法により，Unicode 番号は 0 から 10FFFF となりました。これは 16 ビットを超えるものです。ですから，標準的な方法でこの文字を表すと，2 バイト必要になります。この文字を使用することは，Unicode の最初の目標「固定幅で文字を表す」を修正するものでした。この領域は将来的に極めてまれな文字を割り当てる可能性を残すものとして，文字は割り当てられませんでした。

また，8 ビットエンコーディングとして，UTF-8 が規定されました[*37] 。また，UTF-16 についても言及されています。

纏めると，

- 登録文字数 47,400
- UTF-16 でサロゲートペアの導入
- Unicode 番号は 0 から 10FFFF
- UTF-32 の予感

となります。

[*33] 割り当てられたコードで，必ずしも文字以外のものも含みます。

[*34] バイトを並べる順序のこと。メモリは普通バイト単位で，番号付けされます。例えば，1 番地のバイト数を A，2 番地のバイト数を B とします。これらの 2 バイト (1,2 番地) で数を表す場合，$A \times 256 + B$ とする方法と $A + B \times 256$ とする方法があります。前者を Big-Endian，後者を Little-Endian と言います。

[*35] https://www.unicode.org/versions/Unicode2.0.0/

[*36] Ver. 2 の段階でも未割り当てが 18000 もあり，将来予想される必要数を超えると述べられています。

[*37] UTF は Unicode Transformation Format に由来します。]

4.4 Unicode Ver.3(2000)

2000 年には Ver. 3.0 が発表されました[*38]。Ver. 3 では 57,709 文字が登録されています。

UTF について，明示的に定義がなされ，16 ビット Encoding，UTF-16BE，UTF-16LE の定義が述べられています。

日本語関係では，半角，全角（Halfwidth，Fullwidth）の性質が定義され，半角カタカナや全角アルファベットが登録されています[*39]。

・半角文字： ｱｲｳｴｵ
・全角文字： アイウエオ

纏めると，

- 登録文字数 57,709
- UTF-8，UTF-16 の明確な定義
- 半角，全角の定義

となります。

4.5 Unicode Ver.4(2003) とその後

2003 年には Ver. 4.0 が発表されました[*40]。

Ver. 4.0 では，59,213 文字が登録されています。特に，日本語関係では，JIS X 213（拡張漢字）が登録されています。

また，16 ビットを超える領域の Unicode 番号が実際に割り当てられています。例えば，U+10000 から U+1005D に古代ギリシアで使われた音節文字 Linear B Syllabary が登録されました。

例えば，最初の 10 文字 U+10000 から U+10009 は

𐀀 𐀁 𐀂 𐀃 𐀄 𐀅 𐀆 𐀇 𐀈 𐀉

となります。

このように，Ver. 4.0 では，16 ビットを超えた番号が実際に登録されました。これにより，「16 ビット固定幅で全ての文字を表すという」Unicode の当初の目標は果たされないことになりました。しかしその代わり，32 ビットエンコーディング UTF-32 が規定されました。UTF-32 では，すべての文字が Unicode 番号と直接対応付けられ，これを用いれば，固定幅での表現が得られることになります。

[*38] https://www.unicode.org/versions/Unicode3.0.0/

[*39] Shift-JIS では，半角は 1 バイト，全角は 2 バイトとその文字を保存する際のメモリの量も意味しました。しかし，Unicode では Halfwidth，Fullwidth は文字の幅を示す性質で，その文字を保存するためのメモリを意味するものではありません。

[*40] https://www.unicode.org/versions/Unicode4.0.0/

Ver.1.0 から Ver.4.0 を通じて次が規定されました。

- 登録文字数 59,213
- 8,16,32 ビット用 Encodeing `UTF-8,UTF-16,UTF-32`
- Endian 判定記号 Byte Order Mark `BOM`
- Unicode 番号を `0` から `10FFFF` とする。
- 日本語関係では拡張漢字[*41]を含む

これらの，規定により，Ver. 4.0 で現在のユニコードの基本的枠組みが完成したと言えます。

その後更新が続けられ，最新版は Ver. 15(2022 September 13) です[*42]。149,186 文字が登録されています。Ver.1.0.0 は，全体で 700 ページ弱，Ver.15.0.0 は全体で 1000 ページ強にもなる文書です。，

ユニコードが制定されると，コンピューター内部のコードとしてユニコードが使われるようになりました。現在では，Windows，Mac，Linux，Android 等が採用しています。

日本語 Windows では `Unicode` が採用される前は，内部コードは `Shift-JIS` を使っていました。この状況では，例えば，日本語の文章に「très bien」と言う言葉を含めることはできません。日本語に限ってみても，人名や地名などで使われる特殊な漢字，例えば鴎，鷗や吉や吉についてみると，`Shift-JIS` では鷗や吉を扱うことはできませんが，`Unicode` では扱うことができます[*43]。

4.6 Unicode の成果

ユニコードは世界中の文字を一つの文字集合で表し，それらを簡単に効率的に利用する方法の模索でした。Unicode Ver.1 が発表されて，約 30 年が経ち，その成果を纏めると次のようになります。

- ユニコードは世界中のコンピューターで使われている。⇒ Unicode は世界標準
- ユーザーは，一つのエンコーディングで世界中の文字を混在してコンピューターで使用できる。
- 状況に応じて，`UTF-8`，`UTF-16`，`UTF-32` を使うことができる。これらはすべてユニコードとして等価である。

例えば，Unicode を使うと，

パソコン，personal computer，persönlicher Rechner，Ordinateur personnel，个人电脑，개인용 컴퓨터

のように，外国語が混在する文章を扱うことができます。

[*41] 基本漢字，補助漢字を含みます。
[*42] 2023 年 7 月現在，https://www.unicode.org/versions/Unicode15.0.0/
[*43] 画面等に正しく表示するには対応するフォントが適切にインストールされている必要があります。

第 II 部
ユニコードの仕組み

5　ユニコード

5.1　符号化文字集合

ユニコードは世界中の文字を一つの体系で表そうとしていますから，ユニコードの符号化文字集合は世界中の文字を含んでいます。その結果，かなり大きなものです。欧米で使われているラテン文字はもちろん，日本語の他，ギリシア語，ロシア語，アラビア語，中国語，ハングルなどが含まれています。1991 年にバージョン 1.0.0 が発表された時は，7161 個の文字が登録されていましたが，その後バージョンが上がるにつれて，文字が増え，2006 年にバージョン 5.0 になった時は，99089 文字が登録されました。2023 年 7 月現在，バージョン 15.0 で，149,186 個の文字が登録されています。携帯で使われている絵文字や変体仮名も登録されています。

ユニコードの符号化文字集合は，16 進数で番号付けされています。文字はその番号に U+ を付け加えて表すことになっています。例えば，672C 番目の文字は，U+672C と表します。ユニコードは最初は 16 ビットで番号付けされました。16 ビットは 4 桁の 16 進数ですから，U+0000 から U+FFFF と表されます。

ところで，16 ビットで表される数は，$2^{16} = 65536$ です。上の表示法では，

U+0000　～　U+FFFF

と表されますが，これで世界中のすべての文字を表す構想でしたが，実際には不足でした。そこで，1996 年にユニコードがバージョン 2.0 になった時，この文字が拡張され，U+20B9F のような 16 進 5 桁の文字番号も登録されています。U+0000〜U+FFFF を基本領域（基本多言語面），U+10000 以上を拡張領域と言われています。現在，その上限は U+10FFFF とされていて，最大 1,114,112 個の文字が登録できます。まとめると

ユニコードは U+0000，・・・，U+10FFFF のように番号付けされた多国語文字の集まり。

U+0000 ～ U+FFFF　　を基本領域，
U+10000 ～ U+10FFFF　を拡張領域と言う。

となります[*44]。

例えば，”日”という文字は，26085 番目の文字として登録されています。これを 16 進数として表すと，65E5 ですから，U+65E5 となります。

先頭部分 U+0000 ～ U+007F は Basic Latin の名称で ASCII コードがそのまま割り当てられています。

また，次の U+0080 ～ U+00FF は，Latin-1 Supplement の名称で，ECMA-94（ISO/IEC 8859）Latin 1 の上位 8 ビット部分が割り当てられています。

U+0100 以降，種々の言語の文字が割り当てられていますが，膨大でここで説明するには適していません。

[*44] 基本領域の内 D800〜DFFF はサロゲート領域と言って，実際にはユニコード文字は割り当てられていません。

全体像は，https://unicode.org/charts/で見ることができます。

ここでは，日本語関係について説明します。

日本語関係の文字規格としては，JIS X 0201，JIS X 0208，JIS X 0212，JIS X 0213 があります。これらで登録された文字は，すべて Unicode に登録されています。更にこれ以外では変体仮名のようなものも登録されています。ただ，これらは順次登録されたためユニコード番号としては纏まったブロックとしてあるわけではありません。特に，漢字は中国や韓国の漢字とまとめたグループとして登録されています。

半角カタカナは，Halfwidth Katakana variants と言う名称で U+FF65〜U+FF9F に登録されています。

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
U+FF60						･	ヲ	ァ	ィ	ゥ	ェ	ォ	ャ	ュ	ョ	ッ
U+FF70	ｰ	ｱ	ｲ	ｳ	ｴ	ｵ	ｶ	ｷ	ｸ	ｹ	ｺ	ｻ	ｼ	ｽ	ｾ	ｿ
U+FF80	ﾀ	ﾁ	ﾂ	ﾃ	ﾄ	ﾅ	ﾆ	ﾇ	ﾈ	ﾉ	ﾊ	ﾋ	ﾌ	ﾍ	ﾎ	ﾏ
U+FF90	ﾐ	ﾑ	ﾒ	ﾓ	ﾔ	ﾕ	ﾖ	ﾗ	ﾘ	ﾙ	ﾚ	ﾛ	ﾜ	ﾝ	ﾞ	ﾟ

カタカナ[*45]は Katakana と言う名称で，U+30A0〜U+30FF に登録されています。

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
U+30A0	=	ァ	ア	ィ	イ	ゥ	ウ	ェ	エ	ォ	オ	カ	ガ	キ	ギ	ク
U+30B0	グ	ケ	ゲ	コ	ゴ	サ	ザ	シ	ジ	ス	ズ	セ	ゼ	ソ	ゾ	タ
U+30C0	ダ	チ	ヂ	ッ	ツ	ヅ	テ	デ	ト	ド	ナ	ニ	ヌ	ネ	ノ	ハ
U+30D0	バ	パ	ヒ	ビ	ピ	フ	ブ	プ	ヘ	ベ	ペ	ホ	ボ	ポ	マ	ミ
U+30E0	ム	メ	モ	ャ	ヤ	ュ	ユ	ョ	ヨ	ラ	リ	ル	レ	ロ	ヮ	ワ
U+30F0	ヰ	ヱ	ヲ	ン	ヴ	ヵ	ヶ	ヷ	ヸ	ヹ	ヺ	・	ー	ヽ	ヾ	ヿ

ひらがなは Hiragana と言う名称で，U+3041〜U+309F に登録されています。

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
U+3040		ぁ	あ	ぃ	い	ぅ	う	ぇ	え	ぉ	お	か	が	き	ぎ	く
U+3050	ぐ	け	げ	こ	ご	さ	ざ	し	じ	す	ず	せ	ぜ	そ	ぞ	た
U+3060	だ	ち	ぢ	っ	つ	づ	て	で	と	ど	な	に	ぬ	ね	の	は
U+3070	ば	ぱ	ひ	び	ぴ	ふ	ぶ	ぷ	へ	べ	ぺ	ほ	ぼ	ぽ	ま	み
U+3080	む	め	も	ゃ	や	ゅ	ゆ	ょ	よ	ら	り	る	れ	ろ	ゎ	わ
U+3090	ゐ	ゑ	を	ん	ゔ	ゕ	ゖ		゙	゚		゛	゜	ゝ	ゞ	ゟ

[*45] 全角カタカナのことです。

変体仮名は Hentaigana と言う名称で，U+1B002〜U+1B11E に登録されています。

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
U+1B000			𛀂	𛀃	𛀄	𛀅	𛀆	𛀇	𛀈	𛀉	𛀊	𛀋	𛀌	𛀍	𛀎	𛀏
U+1B010	𛀐	𛀑	𛀒	𛀓	𛀔	𛀕	𛀖	𛀗	𛀘	𛀙	𛀚	𛀛	𛀜	𛀝	𛀞	𛀟
U+1B020	𛀠	𛀡	𛀢	𛀣	𛀤	𛀥	𛀦	𛀧	𛀨	𛀩	𛀪	𛀫	𛀬	𛀭	𛀮	𛀯
• • •																
U+1B100	𛄀	𛄁	𛄂	𛄃	𛄄	𛄅	𛄆	𛄇	𛄈	𛄉	𛄊	𛄋	𛄌	𛄍	𛄎	𛄏
U+1B110	𛄐	𛄑	𛄒	𛄓	𛄔	𛄕	𛄖	𛄗	𛄘	𛄙	𛄚	𛄛	𛄜	𛄝	𛄞	

日本語漢字は CJK というグループとしてまとめられています。CJK は中国，日本，韓国を意味していますので，このグループには，日本語以外の漢字も含まれています。例えば，U+4E00 から最もよく使われる漢字グループ CJK 統合漢字 (CJK Unified Ideographs) が登録されていて，

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
U+4E00	一	丁	丂	七	丄	丅	丆	万	丈	三	上	下	丌	不	与	丐
U+4E10	丑	丒	专	且	丕	世	丗	丘	丙	业	丛	东	丝	丞	丟	
U+4E20	丠	両	丢	丣	两	严	並	丧	丨	丩	个	丫	丬	中	丮	丯
U+4E30	丰	丱	串	丳	临	丵	丶	丷	丸	丹	为	主	丼	丽	举	丿
U+4E40	乀	乁	乂	乃	乄	久	乆	乇	么	义	乊	之	乌	乍	乎	乏
U+4E50	乐	乑	乒	乓	乔	乕	乖	乗	乘	乙	乚	乛	乜	九	乞	也
• • •																

となっています。上にある文字の中には日本語に無い漢字も含まれています。例えば，U+4E06 は 丆 という文字は日本語にはありません。

5.2 ユニコードの文字符号化方式

ユニコードの文字符号化方式は３種類あります。

UTF-8, UTF-16，UTF-32

です[*46]。（ここで UTF は Unicode Transformation Format を意味します。）

それぞれ，UTF-8 は，8 ビットを単位として，使ってユニコード文字を表し，UTF-16 は，16 ビットを単位として，UTF-32 は 32 ビットを使います。

U+10000 以上の文字は，16 ビットでは表せませんから，この場合，UTF-16 では，２組の 16 ビットを使って１文字を表します。ですから，UTF-16 では，一文字を表すに必要なメモリーは固定されていません。16 ビット１個の場合と，16 ビット２個の場合があります。

同様に UFT-8 は，8 ビット１個から４個を使って表します。

ユニコードはもともと，16 ビットで世界中の文字を表す試みとして始まり，後にその不足を補うために，それ以上のビットを使うことになりました。ですから，16 ビットの領域に実は殆んど普通に使う文字は含まれています。ですから，UTF-16 で表した場合殆んどの文字が 16 ビット１個で表されます。逆に，UTF-32 で表した場合，殆んど 32 ビットの半分 16 ビットで間に合うのに，32 ビット１個を使うことになります。UFT-32 が文字の取扱いは便利で簡単ですが，使用するメモリー領域がほぼ倍必要となります。これに対して，UTF-16 は，多少の面倒な部分はありますが，比較的効率的なメモリー使用になります。

このことから，現在のコンピューターでは内部的には UTF-16 を使う OS もあります。実際，WindowsNT 系列では UTF-16 が昔から使われていました。現在普通に使われている Windows 系の OS は，Windows 11 ですがこれは NT 系です。

UTF-8 は１文字を表すのに１～４個の 8 ビットを使う煩雑さはありますが，ASCII との上位互換もあり，ネット上ではよく使われています。また，最近の Linux では，UTF-8 を外部ファイルの標準符号化方式に採用しているものもあります[*47]。

まとめると　ユニコードの文字符号化方式は主として

- UTF-32，UTF-16，UTF-8 の３種類。
- 現在の Windows では内部的には UTF-16 が使われている。
- 外部ファイルの形式としては，UTF-8 が多く使われている。

となります。

[*46] UTF-7 もありますが，特殊な場合に使われます。
[*47] Windows でも最近は UTF-8 を使う場合が多くなっています。

5.3 Endian

上に説明したように同じユニコードでも，符号化方式が 3 種類もありますが，実は実際に使われている符号化形式は，更に，それぞれ，2 種，4 種と言った細分化された形式があります。

それらは特に，16 ビット，32 ビット数のファイルへの格納方式に依存する問題です。

現在使われているコンピュータは 32 ビット或いは 64 ビット CPU が主流ですが，それらでも，メモリやファイルの出入力では 8 ビット (1 バイト) の単位が良く使われています[*48]。実際，例えばメモリアドレスはバイトに対して割り付けられていて，xx 番地のメモリは，対応する xx 番地にあるバイトを示します。

これに対して，例えば，16 ビット数は 2 バイトで構成されますから，16 ビット数をメモリに格納するには，各バイトに分けてそれぞれメモリアドレスを指定します。この場合，標準的な方法として，2 通りが考えられます。

それについて具体的に説明します。16 ビット a 数は，16 進法で表すと，4 桁で，それを

$$a_1, a_2, a_3, a_4 \qquad (0 \leq a_i \leq \mathtt{F})$$

とすると，$a = a_1 16^3 + a_2 16^2 + a_3 16 + a_4$ となります[*49]。それらを構成する，バイト (a_1, a_2) を上位バイト，バイト (a_3, a_4) 下位バイトと言うことにします。

16 ビット a 数を x 番地からメモリに格納する場合，

- x 番地に上位バイト (a_1, a_2)，$x+1$ 番地に下位バイト (a_3, a_4) を格納する。
- x 番地に下位バイト (a_3, a_4)，$x+1$ 番地に上位バイト (a_1, a_2) を格納する。

の 2 通りが考えられます。(1) を Big Endian，(2) を Little Endian と言います。

32 ビットの場合も同様に，32 ビット b 数は，16 進法で表すと，8 桁で，それを

$$b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8 \qquad (0 \leq b_i \leq \mathtt{F})$$

とすると，$b = b_1 16^7 + b_2 16^6 + b_3 16^5 + b_4 16^4 + b_5 16^3 + b_6 16^2 + b_7 16 + b_8$ となります[*50]。それらを構成する，バイト a_i, a_{i+1} を x 番地からメモリに格納する場合，

- x 番地に (a_1, a_2)，$x+1$ 番地に (a_3, a_4)，$x+2$ 番地に (a_5, a_6)，$x+3$ 番地に (a_7, a_8) を格納する。
- x 番地に (a_7, a_8)，$x+1$ 番地に (a_5, a_6)，$x+2$ 番地に (a_3, a_4)，$x+3$ 番地に (a_1, a_2) を格納する。

の 2 通りが考えられます。(1) を Big Endian，(2) を Little Endian と言います。

[*48] 現在でも，コンピュータに搭載されているメモリを測るのに，何 MB とバイトが使われています。
[*49] ここでの，16 は十進法での 16 を示します。
[*50] ここでの，16 は十進法での 16 を示します。

6　UTF の仕組み

ここでは，UTF の符号化について具体的に説明します。

6.1　UTF-32

UTF-32 は 1 ブロック 32 ビットで表現されます。ユニコード番号は U+0〜U+10FFFF で，すべて 32 ビットの範囲内です。ですから，対応は単純に，ユニコード番号を数値で表すだけです。

例 6.1. 「お」と言う字のコード番号は 12362 ですが，16 進で表すと 304A です。ですから，「お」に対応する UTF-32 のコード 32 ビットは 4 個のバイトで表すと

<0000304A>

です。この UTF-32 のコードを外部ファイルに格納する場合は，Endian に応じて，

Big Endian	00 00 30 4A
Little Endian	4A 30 00 00

になります。

6.2　UTF-16

UTF-16 は 1 ブロック 16 ビットで表現されます。基本領域 U+0000〜U+FFFF の範囲の数は 16 ビットで表現されますが，拡張領域 U+10000〜U+10FFFF の範囲の数は 16 ビットでは表現できません。この場合は，2 つの 16 ビットブロックを使って表現します。

正確には，

(1) 基本領域 U+0000〜U+FFFF の範囲で，D800〜DFFF を除くユニコード番号は，そのまま 16 ビット数で表します。ここで，D800〜DFFF はサロゲートコードポイントと言って，ユニコード文字が割り当てられていない領域です。

(2) 拡張領域 U+10000〜U+10FFFF の範囲の数は，(1) でのサロゲートポイントの数を 2 つ使って表します。この対応は，少し複雑ですが，ビットパターンを使って，次のように定義されています。

ユニコード番号	第 1 番目 16 ビット	第 2 番目 16 ビット
000uuuuu xxxxxxxxxxxxxxxx	110110wwwwxxxxxx	110111xxxxxxxxxx

ここで，wwww = uuuuu − 1 です[*51]。コード番号を 21 ビットで表し，上位 5 ビットを uuuuu，下位 16 ビットを xxxxxxxxxxxxxxxx とします。下位 16 ビットを 6 ビット xxxxxx と 10 ビット xxxxxxxxxx に分け，第 1 番目 16 ビットを 110110wwwwxxxxxx とし，上位サロゲート，第 2 番目 16 ビットを 110111xxxxxxxxxx 下位サロゲートとします。これらの組をサロゲートペアと言います。

[*51] uuuuu の最大値は 10000 なので，uuuuu − 1 は最大 4 ビットです。

例 6.2.

(1)「お」と言う字は U+304A です。ですから,「お」に対応する UTF-16 のコードは 2 個のバイトで表すと

<304A>

です。この UTF-16 のコードを外部ファイルに格納する場合は，Endian に応じて，

Big Endian　　30 4A
Little Endian　4A 30

になります。

(2)「臼」と言う字は，ユニコードでは，U+26951 で基本領域にはありませんから，この文字を，UTF-16 を使うと，サロゲートペアで表されます。
16 進数 26951 を 21 ビットで表すと，00010 0110100101010001 です。一方，上の表で uuuuu に対応する部分は，00010 で，wwww = uuuuu - 1 = 00001 ですから，wwww = 0001 となります。対応するサロゲートペアは，

1101100001011010 と 1101110101010001

となります。これを 16 進で表すと，U+D85A と U+DD51 と表せ，いずれもサロゲート領域にあることが分かります。ですから,「臼」に対応する UTF-16 のコードは

<D85A DD51>

です。この UTF-16 のコードを外部ファイルに格納する場合は，Endian に応じて，

Big Endian　　D8 5A DD 51
Little Endian　5A D8 51 DD

になります。

6.3 UTF-8

UTF-8 は 1 ブロック 1 バイト（8 ビット）使って，ユニコードを表現します。ユニコードの番号に応じて，1 バイトから 4 バイトを使います。場合分けは少し多いですが，その対応は比較的単純です。
UTF-8 のユニコード番号との対応は次の表でビットパターンを使って定義されています。

ユニコード番号	第 1 番目バイト	第 2 番目バイト	第 3 番目バイト	第 4 番目バイト
00000000 0xxxxxxx	0xxxxxxx			
00000yyy yyxxxxxx	110yyyyy	10xxxxxx		
zzzzyyyy yyxxxxxx	1110zzzz	10yyyyyy	10xxxxxx	
000uuuuu zzzzyyyy yyxxxxxx	11110uuu	10uuzzzz	10yyyyyy	10xxxxxx

(1) U+0000 から U+007F までの場合は，１バイトでそのまま表現されます。この範囲ではユニコードは ASCII コードと同じ文字が割り振られているので，ASCII ファイルは UTF-8 のファイルとして正しく認識されます。

(2) U+0080 から U+07FF までの場合は，２バイトで表現されます。上の表にあるように，11 ビット yyy yyxxxxxx を上位 5 ビット yyy yy と下位 6 ビット xxxxxx に分け，110yyyyy を１バイト目に割り振り，10xxxxxx を２バイト目に割り振ります。

(3) U+0800 から U+FFFF までの場合は，３バイトで表現されます。上の表にあるように，16 ビット zzzzyyyy yyxxxxxx を上位 4 ビット zzzz と中位 6 ビット yyyy yy，下位 6 ビット xxxxxx に分け，1110zzzz を１バイト目に割り振り，10yyyyyy を２バイト目に割り振り，10xxxxxx を３バイト目に割り振ります。

(4) U+10000 から U+10FFFF までの場合は，４バイトで表現されます。上の表にあるように，21 ビット uuuuu zzzzyyyy yyxxxxxx を最上位 3 ビット uuu と中上位 6 ビット uu zzzz，中下位 6 ビット yyyy yy，最下位 6 ビット xxxxxx に分け，11110uuu を１バイト目に割り振り，10uuzzzz を２バイト目に割り振り，10yyyyyy を３バイト目に割り振り，10xxxxxx を４バイト目に割り振ります。

第１番目バイトの先頭 4 ビットを見ることで，何バイトコードを表すか見ることができます。
また，UTF-8 はバイト単位での格納ですから，Endian の問題はありません。

例 6.3. (1)
例えば，「A」と言う字は，ユニコードでは，U+0041 で１バイトコード

<41>

で表されます。

(2) また，「©」と言う字は，ユニコードでは，U+00A9 で２バイトコードで表されます。A9 は 2 進 11 ビットで表すと 000 10101001 ですから，１バイト目は 11000010，２バイト目は 10101001 になります。これらをそれぞれ，16 進 2 桁で表すと，C2，A9 になります。ですから，「©」と言う字は UTF-8 では，

<C2 A9>

で表されます。

(3)「お」と言う字はユニコードでは，U+304A ですから，３バイトコードで表されます。U+304A は 2 進 16 ビットで表すと 00110000 01001010 となります。上位 4 ビットは，0011，中位 6 ビットは，0000 01，下位 6 ビットは 001010 となります。ですから，１バイト目は 11100011，２バイト目は 10000001，３バイト目は 10001010 になります。これらをそれぞれ，16 進 2 桁で表すと，E3，81，8A になります。ですから，「お」は UTF-8 では，

<E3 81 8A>

で表されます。

(4)「臼」と言う字は, ユニコードでは, U+26951 ですから，４バイトコードで表されます。U+26951 は 2 進 21 ビットで表すと 00010 01101001 01010001 となります。最上位 3 ビットは，000，上中位 6 ビットは，100110，下中位 6 ビットは 100101，最下位 6 ビットは 010001 となります。ですから，１バイト目は 11110000，２バイト目は 10100110，３バイト目は 10100101，４バイト目は 10010001 になります。

これらをそれぞれ，16 進 2 桁で表すと，F0，A6，A5，91 になります。ですから，「臼」は UTF-8 では，

<F0 A6 A5 91>

で表されます。

6.4 BOM

UTF-16，UTF-32 では，Endian により，ファイル等に格納するバイト順が異なります。ですから，処理する場合，そのことを予め知らなければ，正しいユニコードとして解釈することはできません。システムにより，予め指定されていて，その方式で作られていることが分かっていれば，問題はありません。しかし，ユニコードでは，各 UTF-16，32 に対して，Big，Little いずれも許されています。ですから場合によっては，ファイルごとに，Endian の種類を知らせることが必要があるかも知れません。

そのためにファイルの先頭にどのような方式で書かれているかのマーク BOM（**B**yte **O**rder **M**ark）が考え出されました。

6.4.1 UTF-8

UTF-8 は，Endian が関係ないので，BOM は必要ありませんが，ファイルの先頭に

<EF BB BF>

を置くことで，このファイルが UTF-8 で書かれていることを示します。Unicode の規定では，UTF-8 で BOM の付加は必須とされてはいません[*52]。

6.4.2 UTF-16

UTF-16 の規格はユニコード標準では，UTF-16BE，UTF-16LE，UTF-16 と 3 種が規定されています。

- UTF-16BE
  UTF-16BE は big-endian を使う方式です。UTF-16BE では，BOM は使いません。
- UTF-16LE
  UTF-16LE は little-endian を使う方式です。UTF-16LE では，BOM は使いません。
- UTF-16
  UTF-16 は Endian を特に指定しない方式で，BOM で指定します。
  - BOM：<FEFF>の場合，big-endian
  - BOM：<FFFE>の場合，little-endian
  - BOM 無しの場合，big-endian と解釈されます。

[*52] ASCII ファイルとの互換性を重視すれば，BOM は付けないことになります。

6.4.3 UTF-32

UTF-32LE，UTF-32BE，UTF-32 と 3 種が規定されています。

- UTF-32BE
  UTF-32BE は Big Endian を使う方式です。UTF-32BE では，BOM は使いません。
- UTF-32LE
  UTF-32LE は Little Endian を使う方式です。UTF-32LE では，BOM は使いません。
- UTF-32
  UTF-32 は Endian を特に明示しない方式で，BOM で指定します。
  - BOM：`<0000FEFF>` の場合，big-endian
  - BOM：`<FFFE0000>` の場合，little-endian
  - BOM 無しの場合，big-endian と解釈されます。

7　正規化 (normalization)

7.1　結合文字

ユニコードは世界中の文字を表すことを目的としていますので，結合文字という特別な文字の扱いがあります。

例えば，

が

と言う字は，「か」に濁点「゛」という風に言います。つまり「が」は「か」に「゛」で合成されると考えられます。「か」は U+304B ですし，実は，「゛」は U+3099 というユニコード文字です。これらを合わせると，「が」が合成されます。つまりこの場合，

が　＝　か　＋　゛

となります。ここで，「か」を基底文字，「゛」を結合文字，今の場合「が」を合成済み文字といいます。日本語に関連した結合文字は，「゛」や「゜」などがあります。

欧米語では，「´」や「^」などがあります。例えば，「e」は U+0035 で，「´」は U+0301 で，その合成「é」は U+00E9 で

é　＝　e　＋　´

となります。

これはこれで良いのですが，関連して少し困った状況が存在します。実は，例えば，上の「が」は結合文字を使って 2 文字で表すことになりますが，この文字の作り方とは別に「が」という文字がユニコードに登録されています。U+304C は一文字「が」を表します。このように結合文字を使って合成された文字の多くが，一文字として別に登録されています[*53]。つまりそれらは，文字としての表し方が，ユニコードとして 2 種類あることになります。文字列を比較して処理をするときは，基本的にはユニコードのコード番号で処理しますから，この状況では正しく処理されないことになります。

例えば「が」を検索する場合，U+304B と U+3099 として検索すれば，U+304C で表されている「が」は見つからないことになります。

この状況に対応するため，ユニコードでは，正規化（normalize）という概念があります。これは，文字列を表す標準的な形式を定めておき，その形に変換してから処理をするというものです。

[*53] そうでない文字も存在します。例えば，「が」は JIS X 0213 で 1 面 1 区 87 点の 1 文字ですが，ユニコードでは，「か」と「゜」の合成文字で，1 文字としては登録されていません。

7.2 等価性

ユニコード標準では等価性について次の２種を定めています。

- 正準等価 (Canonical equivalence)
  正準等価の文字列とは，同じ意味の文字列で，同じ表示を持つ文字列のことを言います。
- 互換等価 (Compatibility equivalence)
  互換等価の文字列とは，同じ意味の文字列ですが，同じ表示をもつとは限らない文字列のことを言います。

例 7.1.

(1) 次の文字列は正準等値です。

文字列	が	か゛
表示	が	が
コード	<304C>	<304B 3099>
文字列	é	e ´
表示	é	é
コード	<00E9>	<0035 0301>

(2) 次の文字列は正準等値ではありませんが，互換等値です。

文字列	ＡＢＣ	ABC
表示	ＡＢＣ	ABC
コード	<FF21 FF22 FF23>	<0041 0042 0043>
文字列	ℍ	H
表示	ℍ	H
コード	<210D>	<0048>
文字列	⅓	1/3
表示	⅓	1/3
コード	<2153>	<0031 002F 0033>
文字列	㌽	ハ゜ーセント
表示	㌽	パーセント
コード	<332B>	<30CF 309A 30FC 30BB 30F3 30C8>

7.3 正規化

この正規化の形式もまず次の２種があります。

- D 型正規化形式 (Normalization Form D (NFD))
- C 型正規化形式 (Normalization Form C (NFC))

です。

D 型正規化形式は，結合文字などで表されるものを，結合文字を使って分解します。

C 型正規化形式はその文字を結合された等価な文字に合成します。

例えば

元の文章	：	「に」「っ」「ほ」「゜」「ん」「じ」「ん」
D 型正規化形式	：	「に」「っ」「ほ」「゜」「ん」「し」「゛」「ん」
C 型正規化形式	：	「に」「っ」「ぽ」「ん」「じ」「ん」

となります。この他の正規化として

- KD 型正規化形式 (Normalization Form KD (NFKD))
- KC 型正規化形式 (Normalization Form KC (NFKC))

というものもあります。

これは，等価の文字ではなくで互換文字についての正規化です。互換とは，実質同じ意味の文字という意味で,

例えば，日本語では

「１」（全角文字１）と「1」（半角文字１）

などが対応します。

まとめると,

- ユニコードには文字によっては分解や合成がある。
- その為の結合文字がある。（濁点やアクサンなど・・・）
- 文字列の標準形として正規化という概念がある。

となります。

8 フォント

8.1 フォント

ユニコードは世界中の文字を一つの文字集合で記述するものですから，その文字集合は膨大なものです。ユニコード自体は文字に番号付したもので，その内容を記述したファイルは，比較的単純なものです。しかし，そのファイルの中身をエンコーディングで解釈して，コンピューターで実際に画面等に表示するには，一つのハードルがあります。それは，フォントの問題です。

例えば，「春」と言う文字は，U+6625 ですから，UTF-16 形式のファイルの中で見ると，BigEndian なら バイナリとして<66 25>と格納されています。このデータは 2 バイトの情報で単純ですが，実際の「春」と言う文字を表すには U+6625 に対応付けられた字形（グリフイメージ，字としての画像）の情報が必要です。この字形を表現する方法は幾種類かありますが，いずれにしても 2 バイトといった量ではありません。フォントは対応付けデータと字形データを持った一つのまとまりです。

ユニコードは文字を規定していますが，その字形まで細かく規定しているわけではありません。同じ春を表すものでも違うフォントを使うと，文字の形は微妙に異なります。また，同じフォント系でも太さが異なる字形を持つものもあります。

次は同じ文を 3 種のフォントを異なる太さを使って表現したものです。横並びは同じフォントで異なる太さの文字を表しています。

春眠不覚暁　春眠不覚暁　**春眠不覚暁**
春眠不覚暁　**春眠不覚暁**　**春眠不覚暁**
春眠不覚暁　**春眠不覚暁**

上の文は日本語で使われている文字だけで書かれているので，日本語のフォントで表すことができます。しかし，日本語で使われない漢字は，日本語のフォントでは表すことができないことがあります。例えば

図

は，U+56FE で CJK 統合漢字に含まれる文字ですが，日本の文字にはありませんので，対応するフォントが無ければ表示はできません。この文字を表示するには，中国語漢字用フォントが必要になります。対応するフォントが無いと

□

と表示されてしまいます[*54]。またフォントの表示は表示するソフトウエアの機能に依存しますから，仮にフォントがインストールされていても，使用するソフトウエア用に適切な設定をしないと表示できないこともあります。

フォントの形式は色々ありますが，現在では高機能な Open Type 形式が主流になりつつあります。ですから，フォントを導入する場合，可能ならば Open Type 形式を利用するのが良いでしょう。

[*54] この形は tofu と言われることがあります。

8.2 日本語用のフォント

日本語の文章を書く場合，日本語用のフォントを用意する必要があります。ここでは，日本語用のフォントについて説明します。

8.2.1 Windows

現在販売されているコンピューターは，普通に日本語を書くに必要なフォントは標準的にインストールされています。例えば Windows 11 で標準インストールされている日本語フォントは

MS 明朝，MS ゴシック，メイリオ，游明朝，游ゴシック，
BIZ UD 明朝，BIZ UD ゴシック，UD デジタル教科書体

など豊富です。ですから，特別な表現用でなければ，フォントを新しくインストールする必要はありません。次はそれらのいくつかの表示例です。

メイリオ	吾輩は猫である。名前はまだ無い。
游明朝	吾輩は猫である。名前はまだ無い。
BIZ UD 明朝	吾輩は猫である。名前はまだ無い。
UD デジタル教科書体	吾輩は猫である。名前はまだ無い。

Windows 以外の OS でも，標準的な日本語環境は問題なく利用できるでしょう。

8.2.2 IBM Plex フォント

OS に含まれていなくても，フリーに使用できるフォントもあります。

IBM Plex フォントはその一つです。IBM Plex は，IBM での標準フォント Helvetica に代わる欧文フォントとして開発されました。「個性的でありながら、時代に左右されない。IBM のブランド精神、信念、デザイン原則を反映したものとして設計された」とホームページで述べられています[*55]。

元々は欧文用[*56]ですが，日本語用，IBM Plex Sans JP が公開されています[*57]。次はその表示例です。

IBM Plex Sans JP　吾輩は猫である。名前はまだ無い。

8.2.3 源ノフォント

Pan-CJK（汎-中日韓）の開発を目的とした，Adobe と Google の共同プロジェクトが 2012 年にスタートしました。

その成果として，2014 年に，源ノ角ゴシック，Noto Sans CJK，2017 年に源ノ明朝，Noto Serif CJK の初版がリリースされました。これらは現在も更新が続いています。[*58]

[*55] https://www.ibm.com/blogs/think/jp-ja/how-ibm-plex-an-opensource-font-was-born/

[*56] IBM Plex は，Sans，Mono，Serif，Condensed の 4 種類のフォント・ファミリーでローマンとイタリックの 2 つのスタイルがあります。

[*57] https://github.com/IBM/plex/tree/master/IBM-Plex-Sans-JP

[*58] 源ノ角ゴシック，源ノ明朝は，https://github.com/adobe-fonts から，それぞれ，source-han-sans，source-han-serif としてダウンロードできます。また，Noto Sans CJK，Noto Serif CJK は，https://github.com/notofonts/noto-cjk からダウンロードできます。

これらのフォントは Pan-CJK の名称の通り，日本語は勿論，中日韓３国の文字記号が含まれています[*59]。
以下はその表示例です。

源ノ角ゴシック　**吾輩は猫である。名前はまだ無い。**
源ノ明朝　吾輩は猫である。名前はまだ無い。

8.2.4 花園フォント

花園明朝は花園大学国際禅学研究所で 2007 年に公開された花園明朝が始まりです。現在は，GlyphWiki[*60]で開発されています。
日本語に限らず多くの文字が登録されていることが特徴です[*61][*62]。
以下はその表示例です。

花園明朝　吾輩は猫である。名前はまだ無い。

[*59] 簡体字，繁体字，ハングルも含みます。
[*60] http://fonts.jp/hanazono/，大東文化大学教員上地宏一氏により運営されています。
[*61] https://ja.osdn.net/projects/hanazono-font/releases/ からダウンロードできます。
[*62] AFDKO tool を使って生成した花園明朝 HanaMinAFDKO もあります。https://github.com/cjkvi/HanaMinAFDKO/releases/ からダウンロードできます。

8.3 世界のフォント

8.3.1 GNU Unifont，Arial Unicode MS

ユニコードは世界中の文字を一つの文字集合で表すプロジェクトですから，それと同様に，一つのフォントでユニコードの文字を表すことができれば大変便利です。

「GNU Unifont [*63]」はそのような目的のためのプロジェクトです。また，似たものとして，「Arial Unicode MS [*64]」がありました。

しかし，残念なことにユニコードの文字集合は元々大きく，また年々大きくなっていくためこのようなプロジェクトは中々困難です。以下は Unifont と Arial Unicode MS を使った例です。

Unifont

Hello world!，Γ ε ι α σ α ς κ ό σ μ ο !，¡Hola, mundo，Hallo Welt!，
П р и в е т мир，안녕하십니까 세계!，世界，你好！，こんにちは世界!

Arial Unicode MS

Hello world! , Γ ε ι α σ α ς κ ό σ μ ο ! , ¡Hola, mundo , Hallo Welt! ,
П р и в е т мир , 안녕하십니까 세계! , 世界 , 你好！ , こんにちは世界!

これらのフォントはかなり多くの文字を表示可能ですが，それでもすべてではありません。また，ユニコードの文字集合は元々大きく，また年々大きくなっていきます。それらに含まれる色々な国々の文字の複数書体のサポート，多くの要望に応え，それらを一つのファイルで行うのは大変なことです。そのようなこともあり，現在ユニコードのフォントでは，地域の文字ごとに複数のフォントを用意し，必要な部分だけダウンロードして使用するのが主流です。

8.3.2 Windows

Windows では，多くのフォントが標準で用意されていますから，欧米系の文字であれば特に新たにインストールしなくても間に合うと思います。不足の場合は，「C:¥Windows¥Fonts」のエクスプローラーでフォント追加が可能です。「すべての言語のフォントをダウンロード」を行えば必要なものが揃うと思います。

以下は，Windows での Serif 系フォントを使って表示したものです。

Hello world!，Γ ε ι α σ α ς κ ό σ μ ο !，¡Hola, mundo，Hallo Welt!，
П р и в е т мир，안녕하십니까 세계!，世界，你好！，こんにちは世界!

[*63] https://unifoundry.com/unifont/

[*64] Arial Unicode MS は以前 Office に同梱されていたものですが，2000 年頃にサポートが停止され，以後暫くフリーに配布されていたものでが，現在はその配布も停止されています。

8.3.3 Noto フォント

また，Windows 標準以外では，Google が開発している Noto フォントファミリー[*65]があります[*66]。このフォントは非常に多くの書体をサポートしています。フリーで使用できます。

Noto Sans font

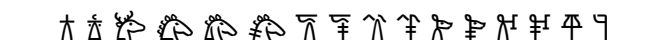
Hello world!, Γ ε ι α σ α ς κ ό σ μ ο !, ¡Hola, mundo, Hallo Welt!,
П р и в е т мир , 안녕하십니까 세계!, 世界，你好！，こんにちは世界!

Noto Serif font

Hello world!, Γ ε ι α σ α ς κ ό σ μ ο !, ¡Hola, mundo, Hallo Welt!,
П р и в е т мир , 안녕하십니까 세계!, 世界，你好！，こんにちは世界!

このような表示が可能です。この場合，Sans，Serf ともに 4 種類のフォントを切り替えて表示しています。また，Noto にはミケーネ線文字や古代エジプトの象形文字といった古代文字もあります。

Noto Sans Linear B



Noto Sans Egyptian Hieroglyphs

Google の Font ページ[*67]では，Noto フォントに限らず，数多くのフォントが公開されています。

8.3.4 花園明朝

花園明朝は，日本語に限らず多くの文字記号をサポートしています。公式ホームページでは，HanaminA および HanaminB と 2 種のフォントが公開されています。HanaMinAFDKO では，HanaminC も公開されています。

HanaminA だけでも次のような表示が可能です。

HanaminA

Hello world!, Γ ε ι α σ α ς κ ό σ μ ο !, ¡Hola, mundo, Hallo Welt!,
П р и в е т мир , 안녕하십니까 세계!, 世界，你好！，こんにちは世界!

[*65] Noto の名称について，ホームページには”Noto” means ”I write, I mark, I note” in Latin. The name is also short for ”no tofu”, as the project aims to eliminate ’tofu’: blank rectangles shown when no font is available for your text. とあります。

[*66] https://fonts.google.com/noto

[*67] https://fonts.google.com/

9　まとめ

現在の私たちは，一つの国に閉じて生活をしている訳ではなく，世界中の多くの国々と交流をして生活をしています。そしてこの傾向は進むことはあれ，後退することはないでしょう。ですから，色々な事柄について，各国の特徴を尊重しながら，それらを統一的な方法で扱うことができれば，大変便利で効率的でしょう。コンピューターでの文字の取扱いについての，その試みの一つがユニコードです。歴史的な状況と，各国々の事情により，ユニコードは種々の問題を含んでいますが，このような試みは歴史の進化に沿ったものであることは確かです。またユニコードは現在のコンピューターの OS の中に随分以前から取り入れられています。最近，それが OS だけではなく，応用ソフトも対応が進みつつあり，そして我々は知らず知らずのうちに，ユニコードを利用しています。

このような中で，現在のコンピューターをより良く利用するためには，ユニコードの理解は避けて通れない事柄でしょう。

まとめると，

- ユニコードはコンピューター進化の方向に沿った試み。
- 一般のコンピューター利用者もこれからは，ユニコードの理解を避けて通れない。

となります。

「ユニコード (Unicode) へ」更新記録

- （2023 年 08 月版）23，26 ページの誤植の訂正。
- （2023 年 07 月版）内容大幅追加・再構成。名称を「ユニコード (Unicode)」から「ユニコード (Unicode) へ」変更。ユニコード以前から，ユニコードに至る経緯，ユニコードの技術についての内容を大幅に追加した。ページ数 7 ページから 51 ページへ。
- （2013 年 10 月版）初版公開