US-ASCII

US-ASCII est un codage d'un jeu de caractères contenant :
— 32 caractères de contrôles d'un télétype,
— les lettres latines,
— les chiffres indo-arabes,
— des signes de ponctuation utiles en anglais.
Il utilise les 7 bits de poids les plus faibles d'un octet pour coder la position d'un caractère dans ce jeu

00	NUL	10	DLE	20	SP	30	0	40	@	50	P	60	`	70	p
01	SOH	11	DC1	21	!	31	1	41	A	51	Q	61	a	71	q
02	STX	12	DC2	22	"	32	2	42	B	52	R	62	b	72	r
03	ETX	13	DC3	23	#	33	3	43	C	53	S	63	c	73	s
04	EOT	14	DC4	24	$	34	4	44	D	54	T	64	d	74	t
05	ENQ	15	NAK	25	%	35	5	45	E	55	U	65	e	75	u
06	ACK	16	SYN	26	&	36	6	46	F	56	V	66	f	76	v
07	BEL	17	ETB	27	'	37	7	47	G	57	W	67	g	77	w
08	BS	18	CAN	28	(	38	8	48	H	58	X	68	h	78	x
09	HT	19	EM	29	)	39	9	49	I	59	Y	69	i	79	y
0a	LF	1a	SUB	2a	*	3a	:	4a	J	5a	Z	6a	j	7a	z
0b	VT	1b	ESC	2b	+	3b	;	4b	K	5b	[	6b	k	7b	{
0c	NP	1c	FS	2c	,	3c	<	4c	L	5c	\	6c	l	7c	|
0d	CR	1d	GS	2d	-	3d	=	4d	M	5d	]	6d	m	7d	}
0e	SO	1e	RS	2e	.	3e	>	4e	N	5e	^	6e	n	7e	~
0f	SI	1f	US	2f	/	3f	?	4f	O	5f	_	6f	o	7f	DEL

Unicode et UTF-8

La norme ISO/CEI 10646 définit un jeu universel de caractères (**Universal Character Set** ou UCS en anglais) pour tous les systèmes d'écriture de la planète. C'est la base de la norme **Unicode**[1] gérée par le consortium Unicode. C'est une **liste ordonnée** de noms de caractères : chaque caractère y est repéré par une **position** (*code point* en anglais).

UCS définit 2 097 152 positions, mais ne définit pas de codage des nombres représentant ces positions. On fait référence aux caractères d'UCS par leur position via la notation **`U+XXXX`** où `XXXX` représente un nombre de 4 à 6 chiffres hexadécimaux.

UTF-8 (*UCS Transformation Format 8 bits*) est un format de codage des positions du jeu de caractère universel, crée par Ken Thompson et Rob Pike. C'est une façon de transformer une position dans UCS, c'est-à-dire un numéro, en une séquence de 1 à 4 octets.

Ce codage simple reprend le codage US-ASCII (avec un bit de poids fort à 0) pour toutes les positions `U+0000` à `U+007F`, puis pour les autres positions :
— le premier octet est construit en positionnant sur les bits de poids forts autant de 1 que d'octets nécessaires au codage de la position, suivi d'un 0 ;
— en positionnant les 2 bits de poids forts de chacun des octets suivants à `10` ;
— les bits **utiles**, les bits à coder, sont alors répartis sur les bits libres restants.

Valeurs	bits utiles	Position Unicode en binaire	Codage UTF-8 en binaire	octets
U+0000 → U+007F	7	abc defg	**0**abc defg	1
U+0080 → U+07FF	11	abc defg hijk	**110**a bcde **10**fg hijk	2
U+0800 → U+FFFF	16	abcd efgh ijkl mnop	**1110** abcd **10**ef ghij **10**kl mnop	3
U+10000 → U+1FFFFF	21	a bcde fghi jklm nopq rstu	**1111 0**abc **10**de fghi **10**jk lmno **10**pq rstu	4

1. Unicode étend le jeu de caractères en définissant plus d'informations que le simple nom du caractère. Il va donc plus loin que ISO/CEI 10646 en ce sens que c'est plus qu'un simple jeu de caractères abstrait.

Encodage UTF-8

Pour encoder une position Unicode en UTF-8 :

1. Considérer la ligne correspondant à votre position Unicode
2. Convertir votre position Unicode en binaire
3. Garder les 7, 11, 16 ou 21 derniers bits (dépendant de la ligne utilisée)
4. Identifier chacun des bits conservés avec les lettres de la colonne <Position Unicode en binaire> (chaque lettre correspondant à exactement 1 bit)
5. Écrivez votre code UTF-8 sur 1, 2, 3, ou 4 octets (dépendant de la ligne utilisée) en suivant la forme donnée par la colonne <Codage UTF-8 en binaire>

Décodage UTF-8

Pour obtenir une position Unicode à partir d'une séquence d'octets encodés en UTF-8 :

1. Écrire la séquence d'octets en binaire.
2. Regarder le nombre de `1` à la suite parmi les bits les plus à gauche

Nombre de `1`	0	2	3	4
Encodage sur	1 octet	2 octets	3 octets	4 octets

3. Considérer le nombre d'octets trouvé précédemment
4. Identifier chaque bit avec les chiffres ou les lettres de la colonne <Codage UTF-8 en binaire> (chaque lettre correspondant à exactement 1 bit, les chiffres doivent correspondre)
5. Écrivez votre position Unicode en binaire en suivant la forme donnée par la colonne <Position Unicode en binaire>
6. Convertissez en hexadécimal pour avoir une position Unicode de la forme `U+XXXX` ou `U+XXXXXX`.

Exercices

Exercice 1 : Code ASCII

Q1. En vous aidant de la table de conversion ASCII, déterminez :
- le code du caractère `a` et celui du caractère `A`
- l'écriture binaire des deux codes précédents
- le code du caractère de *saut de ligne*
- le caractère de code `0x30`

Q2. Comment passe-t-on du code d'une lettre latine au code de la lettre suivante ?

Q3. Comment passe-t-on du code d'une capitale à sa version minuscule pour les lettres latines ?

Exercice 2 : UTF-8

Q1. Quel est le codage UTF-8 des caractères suivants dans le jeu UCS :
- caractère `LATIN CAPITAL LETTER A`, représenté par `A`, à la position `U+0041` ?
- caractère `LATIN SMALL LETTER E WITH ACUTE`, représenté par `é`, à la position `U+00E9` ?
- caractère `LATIN SMALL LETTER AE`, représenté par `æ` à la position `U+00E6` ?
- caractère `AIRPLANE`, représenté par ✈, à la position `U+2708` ?

Q2. Dans un fichier codé en UTF-8, on trouve les six octets suivants `0xE6 0x9D 0x8c 0xDE 0xBC 0x43`.
- Combien de caractères sont réellement codés dans ce texte ?
- Quelles sont les positions de ces caractères dans UCS ?

Exercice 3 : Base 64

Le codage base64 est un codage permettant de transformer toute donnée binaire en une donnée n'utilisant que 64 caractères ASCII. Il permet, par exemple, de transmettre n'importe quelle donnée binaire par les protocoles de communication n'autorisant que les caractères ASCII (comme le courrier électronique).

Le principe de ce codage consiste à découper la donnée binaire en tranches de six bits, appelées *sextets*, et d'associer à chaque sextet un caractère choisi parmi les 26 lettres latines capitales et minuscules, les 10 chiffres arabes et les deux caractères `+` et `/`.

La table de codage est la suivante :

Sextet (déc.)	Code	**Sextet (déc.)**	Code	**Sextet (déc.)**	Code	**Sextet (déc.)**	Code
`000000` (0)	`A`	`010000` (16)	`Q`	`100000` (32)	`g`	`110000` (48)	`w`
`000001` (1)	`B`	`010001` (17)	`R`	`100001` (33)	`h`	`110001` (49)	`x`
`000010` (2)	`C`	`010010` (18)	`S`	`100010` (34)	`i`	`110010` (50)	`y`
`000011` (3)	`D`	`010011` (19)	`T`	`100011` (35)	`j`	`110011` (51)	`z`
`000100` (4)	`E`	`010100` (20)	`U`	`100100` (36)	`k`	`110100` (52)	`0`
`000101` (5)	`F`	`010101` (21)	`V`	`100101` (37)	`l`	`110101` (53)	`1`
`000110` (6)	`G`	`010110` (22)	`W`	`100110` (38)	`m`	`110110` (54)	`2`
`000111` (7)	`H`	`010111` (23)	`X`	`100111` (39)	`n`	`110111` (55)	`3`
`001000` (8)	`I`	`011000` (24)	`Y`	`101000` (40)	`o`	`111000` (56)	`4`
`001001` (9)	`J`	`011001` (25)	`Z`	`101001` (41)	`p`	`111001` (57)	`5`
`001010` (10)	`K`	`011010` (26)	`a`	`101010` (42)	`q`	`111010` (58)	`6`
`001011` (11)	`L`	`011011` (27)	`b`	`101011` (43)	`r`	`111011` (59)	`7`
`001100` (12)	`M`	`011100` (28)	`c`	`101100` (44)	`s`	`111100` (60)	`8`
`001101` (13)	`N`	`011101` (29)	`d`	`101101` (45)	`t`	`111101` (61)	`9`
`001110` (14)	`O`	`011110` (30)	`e`	`101110` (46)	`u`	`111110` (62)	`+`
`001111` (15)	`P`	`011111` (31)	`f`	`101111` (47)	`v`	`111111` (63)	`/`

Q1. Quel est le codage en base64 des 3 octets `0x20 0x42 0x08` ?

Q2. Quels sont, exprimés en binaire et en hexadécimal, les octets codés en base 64 par `ZMOpY29kYWdl` ?

Q3. Cette chaine représente un mot lui-même codé en UTF-8. Quel est ce mot ?