SÉANCE PROGRAMMES : PROGRAMMATION ET SCRIPTS

Objectif : L'objectif de la séance est de mobiliser la pensée algorithmique au travers de la production d'algorithmes simples pour comprendre le monde qui nous entoure et ses évolutions dans une perspective d'émancipation, et d'automatiser des tâches. Cet objectif passe par le rappel des codages les plus utilisés. Des exemples seront vus dans les algorithmes et les pages HTML.

A - Quelques définitions

Algorithme : est, au sens propre, l'ensemble des règles opératoires intervenant dans toute forme de calcul. Par extension il signifie le mécanisme réglant le fonctionnement de la pense organisée et s'explicitant par des représentations mathématiques ou logiques.

Programme : au sens propre, l'ensemble des actions qu'on se propose d'accomplir dans un but déterminé. Par extension : liste des instructions écrites sous forme conventionnelle qui permettent l'exécution d'un travail sur une machine.

B - Représentation et codage de l'information

L'information manipulée par un ordinateur est essentiellement électrique. La plupart des valeurs traitées sont représentées sous forme numérisée et codée. Au plus bas niveau en binaire, c'est à dire en base 2 : que ce soient les caractères affichés sur un écran ou la mesure du déplacement de la souris.
Dans la vie de tous les jours nous sommes habitués à plusieurs bases de numération : la base décimale (base 10) pour le comptage courant, la base sexagésimale (base 60) pour les heures, mais aussi en base 24 pour les jours, en base 365 pour les années, en base 360 pour les angles en degrés, etc.

En base 2, les seuls chiffres utilisés sont 0 et 1 ce qui peut être fastidieux à écrire comme à lire.

Déc.	binaire	Hexa.		Déc.	binaire	Hexa.
0	0 0000	00		16	1 0000	10
1	0 0001	01		17	1 0001	11
2	0 0010	02		18	1 0010	12
3	0 0011	03		19	1 0011	13
4	0 0100	04		20	1 0100	14
5	0 0101	05		21	1 0101	15
6	0 0110	06		22	1 0110	16
7	0 0111	07		23	1 0111	17
8	0 1000	08		24	1 1000	18
9	0 1001	09		25	1 1001	19

10	0 1010	0A	26	1 1010	1A
11	0 1011	0B	27	1 1011	1B
12	0 1100	0C	28	1 1100	1C
13	0 1101	0D	29	1 1101	1D
14	0 1110	0E	30	1 1110	1E
15	0 1111	0F	31	1 1111	1F

D'autres systèmes ont été utilisés au fil du temps comme le système octal (en base 8 et qui utilise les chiffres de 0 à 7) ou plus couramment hexadécimal (en base 16 qui utilise tous les chiffres plus les lettres de A à F).

L'unité de base étant un nombre binaire (*binary digit* en anglais) ou **bit**. Les valeurs possibles d'un bit sont donc 0 ou 1, soient 2 valeurs.

La valeur 11 en décimal s'écrira 1011 en binaire.
Il est important de retenir qu'il s'agit de la même valeur, seule la représentation, l'écriture, change.

En rappel, la construction de l'écriture en base suit toujours le même schéma de calcul :
exemple : en base N, le nombre 1234 a comme valeur numérique

$$1234_N = 1_{10}xN^3 + 2_{10}xN^2 + 3_{10}xN^1 + 4_{10}xN^0$$

Afin d'éviter les confusions, la valeur de la base n'étant pas systématiquement indiquée, on a ajouté des caractères supplémentaire :

ou 0x ou $ ou H précédant un nombre indique la base hexadécimale
O ou \0 indique une valeur écrite en octal
Il y a cependant des cas particuliers (voir plus loin).

Le traitement de valeurs numériques représentant d'autres objets que des valeurs comptables reste cependant une grande part des traitement informatiques. Il s'agit principalement des adresses et des caractères d'écriture (ainsi que des codes de contrôle de l'édition comme par exemple le saut de ligne).

La plupart des ordinateurs, des systèmes et des programmes standardisent le nombre de bits pour simplifier aussi bien le matériel (hardware, les interfaces etc.) que les logiciels.
Ces standards évoluent avec le temps et l'intégration de l'électronique.
On appelle **mot** le groupement de bits traité par une machine selon sa conception.
Si le premier processeur avait des mots de 4 bits, plusieurs décennies d'ordinateurs ont utilisé des mots de 8 bits (et des multiples de 8 bits).
On appelle **octet** un mot de 8 bits (symbole « o ») ou « **Byte** » (symbole B) en anglais.
L'usage et l'habitude fait que le jargon informatique continue d'utiliser le mot octet pour spécifier certaines données informatiques indépendamment de la longueur des mots machine : les capacités de stockages sont exprimées en octets (et ses multiples).

Noter que traditionnellement l'informatique utilise des puissances de 2 comme unité de base. Jusqu'à une époque assez récente, 1 Mo signifiait 1 méga-octet et avait pour valeur 1024x1024 octets. Or ceci n'était pas conforme au SI (Système International de notation). Afin d'éviter les confusions de nouvelles utilités non-SI ont été introduites en insérant un « i » entre le multiple et le « o » de l'octet. Cette double notation prête souvent à confusion, notamment lors de l'indication de capacités de stockages des disques par les fabricants.

Préfixe usuel	Facteur de 10 (10^x)	Abrégé base 10 (facteur 1000)	Abrégé base 2^{10} (facteur 1024)	Nombre de bits (2^x)
octet	0	o	o	8
kilo-octet	3	ko	kio	10
méga-octet	6	Mo	Mio	20
giga-octet	9	Go	Gio	30
tera-octet	12	To = 1000 Go	Tio	40
péta-octet	15	Po = 1000 To	Pio	50
exa-octet	18	Eo = 1000 Po	Eio	60
zetta-octet	21	Zo = 1000 Eo	Zio	70
yotta-octet	24	Yo = 1000 Zo	Yio	80

Les ordinateurs qui sortent sur le marché actuellement sont quasiment tous avec des mots de 64 bits. Les systèmes embarqués ont souvent des processeurs de 8 ou 16 bits.

LE CODAGE DES CARACTÈRES

Le code de base est le code ASCII (American Standard Code for Information Interchange) qui a été augmenté au fur et à mesure des besoins : internationalisation et progrès des traitements de texte, en particulier pour adapter la numérisation de l'édition dans les langues avec accents (Français, langues nordiques, etc.) puis aux langues à alphabets plus complexes (idéogrammes : le chinois, le japonais, le sanscrit, les langues orientales, etc.)

Pour des raisons de compatibilité ascendante, la plupart des codages récents sont compatibles avec les 128 caractères du codage ASCII.

Les codages les plus utilisés pour l'écriture sont maintenant par ordre d'utilisation :

- les codages UTF (Universal coded character set Transformation Format) ou Unicode standard, en 3 formats : UTF-8 (8 bits), UTF-16 (16 bits) et UTF-32 (32 bits) (cf. ISO/IEC 10646)
- ISO-8859 : norme déclinée selon les classes de dialectes. L'ISO-8859-15 (code Latin-1 + sigle Euro) correspond aux accents français.
- ASCII

Le standard actuel est l'UTF-8 (décliné par pays : UTF-8_FR pour les caractères utilisés en français). Le plus versatile UTF-32 est censé pouvoir être utilisé pour toutes les langues et dialectes

humains passés … et futurs !

Voir en annexe le codage ASCII et un exemple d'UTF.

LE CODAGE DES DONNÉES

Les données traitées peuvent être multi-formes (mélange de texte, de valeurs, de références etc.)
Les données purement numériques : entiers et décimaux.
Les entiers sont stockés sous forme d'octets dépendant de la capacité des machines : 1, 2, 4 ou 8 octets.
Les entiers signés se décomposent en :

- 1 bit de signe (0 pour positif, 1 pour négatif)
- (N-1) bits pour la valeur

Exemple : sur un octet (256 valeurs de 0 à 255) on obtiendra :

- non-signé : de 0 à 255
- signé : de -128 à +127

Lorsque la valeur entière est stockée sur plus d'un un octet, par exemple sur 16 bits, un des octets constitue la partie de faible valeur de l'entier (2^0 à 2^8) ou « poids faible » et l'autre octet la partie de plus forte valeur (2^9 à 2^{15}) ou « poids fort ».

Tous les constructeurs informatiques n'ont bien sûr pas adopté la même convention. C'est pourquoi vous trouverez dans la littérature deux types de machines selon que l'octet de poids fort est stocké en premier ou en dernier.

En anglais on parle de « **little endian** » pour ceux dont le poids fort (MSB : most significant Byte) est en premier et « **big endian** » pour ceux dont le poids faible (LSB : less significant Byte) est en premier. Ces différences sont connues et intégrées dans les interfaces afin que des ordinateurs des deux mondes puissent communiquer.

Cependant, si vous échangez des fichiers binaires contenant des données numériques, il conviendra de vérifier la concordance de la machine qui traite le fichier avec celle qui l'a créé sous peine d'obtenir des données inutilisables sans pour autant que la machine ne détecte d'erreur !

Les formats plus évolués notamment pour les décimaux le standard **IEEE-754** décrit une manière classique de les encoder.
Cet encodage permet d'écrire des valeurs très grandes et très petites, ce qui peut induire des erreurs de calcul.

C - Pensée algorithmique

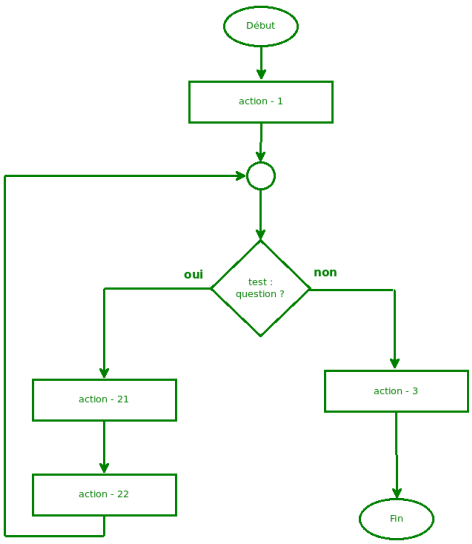
Le déroulement des actions ayant pour objectif d'accomplir une certaine tâche peut être décrit ou décomposé en une succession d'actions plus élémentaires à exécuter séquentiellement ou parallèlement.

Un algorithme constitue la formulation de la logique qui sous-tend cette description.
La plupart des algorithmes peuvent être formulés sous une forme plus ou moins standardisée mais globalement sous deux formes :

- description graphique

- description symbolique

DESCRIPTION GRAPHIQUE D'UN ALGORITHME



La description graphique d'un algorithme est assez trivial à lire. Les flèches indiquent le sens d'exécution. Le format des « boîtes » est généralement lisible sans nécessité de documentation.

Certaines normes de descriptions sont cependant plus complexes. Par exemple, le Grafcet (Graphe Fonctionnel de Commande des Etapes et Transitions) pour les automatismes ou l'UML (Unified Modeling Language) pour la conception de systèmes utilisent des formes de « boîtes » ou de flèches portant une signification particulière.

DESCRIPTION SYMBOLIQUE D'UN ALGORITHME

Plusieurs standard sont concurrents dans la littérature, Donald Knuth a décrit une syntaxe généralement utilisée (avec plus ou moins de rigueur).

On y retrouve les éléments de base d'un algorithme :

- l'affectation d'une valeur à une variable (v ← 2)
- l'appel d'une fonction ou d'un sous-programme (lire())
- les tests conditionnels (si … alors … sinon ...)
- boucles (faire tant que ...)

Exemple :

```
i ← 0 ;
a ← Lire() ;
Tant que a ∓ 2 faire :
    i ← i + 1 ;
    a ← Lire() ;
Ecrire(i) ;
```

COMPLEXITÉ D'UN ALGORITHME

La complexité d'un algorithme s'évalue généralement à partir du lien entre le temps d'exécution et la quantité d'information à traiter. Elle se note O(…).
Par exemple, on trouvera :

- O(1) complexité indépendante de la taille des données
- $O(n^2)$ ou complexité quadratique (par exemple un produit cartésien d'ordre n)
- O(nLog(n)) complexité quasi-linéaire (par exemple le tri *quick sort* de n valeurs)
- $O(n^p)$ complexité polynomiale
- $O(2^n)$ complexité exponentielle

- etc.

D - Programmation

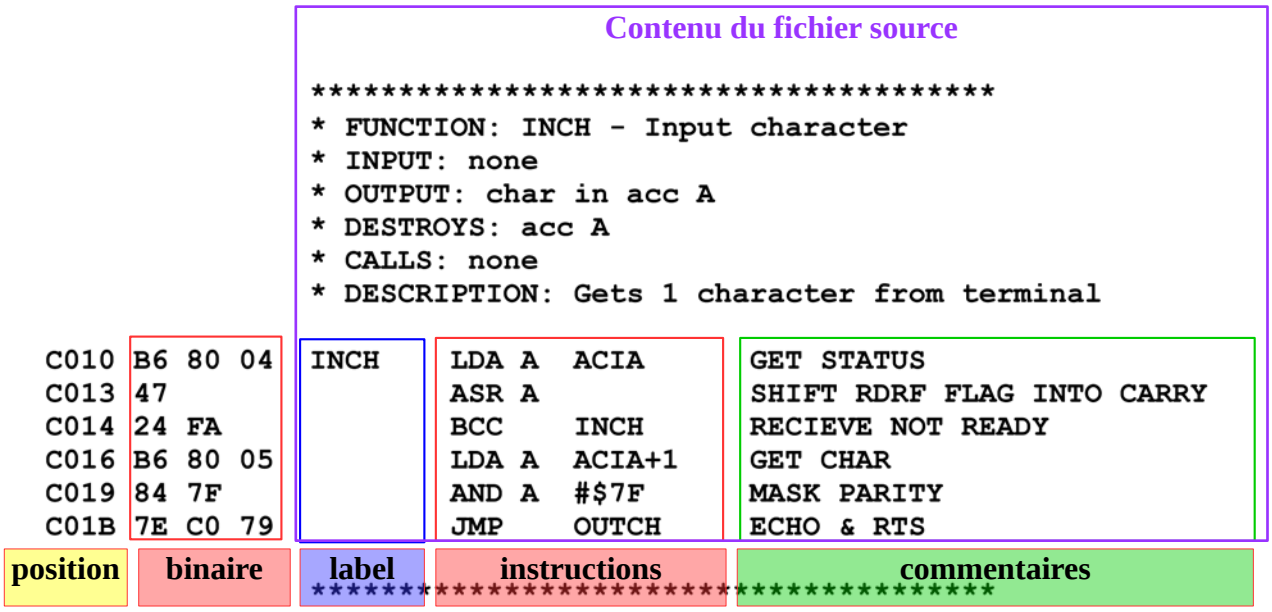
La machine ne sait traiter que du binaire : les valeurs et les instructions doivent être traduites.

EVOLUTION DES LANGAGES

Les instructions processeur sont encodées en binaire
L'hexadécimal est une manière alternative de représenter ces instructions mais reste encore difficile à interpréter pour les non initiés, d'autant moins que le binaire machine dépend non seulement du processeur mais aussi des périphériques qui lui sont associés.

Une approche plus lisible a été le code **assembleur symbolique**. Les instructions sont traduites en fonctions élémentaires abrégées sur 3 ou 4 caractères associées à des modes d'adressage pour récupérer les données à traiter, c'est le **code source**. Pour traduire ce langage symbolique en binaire, le code source doit être traduit en binaire compréhensible par la machine. Pour ce faire on utilise un programme appelé **assembleur**. Le langage assembleur symbolique, déjà beaucoup plus compréhensible que le binaire pur, est toujours attaché à une machine particulière (processeur et implémentation).

Exemple de code assembleur Motorolla 6800 :



Avec la notion de **portabilité** d'un programme, l'écriture du code est passé à un niveau d'abstraction supplémentaire : ce sont les langages de haut niveau dont l'emblématique langage C. Il n'en demeure pas moins que ces langages doivent être transcrits en binaire compréhensibles par la machine pour pouvoir être exécutés.
Deux cas (et demie) se profilent :

- les langages compilés
- les langages interprétés

LES LANGAGES COMPILÉS

Les **langages compilés** (comme le C, le C++, etc.) subissent une série de transformation pour aboutir à un **exécutable** (fichier contenant du code binaire machine). Deux étapes importantes se retrouvent dans la plupart des processus :

- *la compilation* : traduction du **code source** en **code objet**
- *l'édition de liens* : mise en commun de plusieurs codes objet pour aboutir au code machine final

La seconde étape permet la réutilisation de parties de programmes (les librairies) sans avoir besoin de tout réécrire.

LES LANGAGES INTERPRÉTÉS, LES SCRIPTS

Les **langages interprétés** sont avalés par un **interpréteur** qui va traduire les code source au fur et à mesure (ligne à ligne) et l'exécuter après chaque traduction sans générer de fichier binaire.
Cette traduction « au vol » est à la fois une source de flexibilité et un inconvénient majeur en terme de performance.
Les interpréteurs Shell (sh, csh, tcsh, ksh, bash, etc.) sont les plus courants.

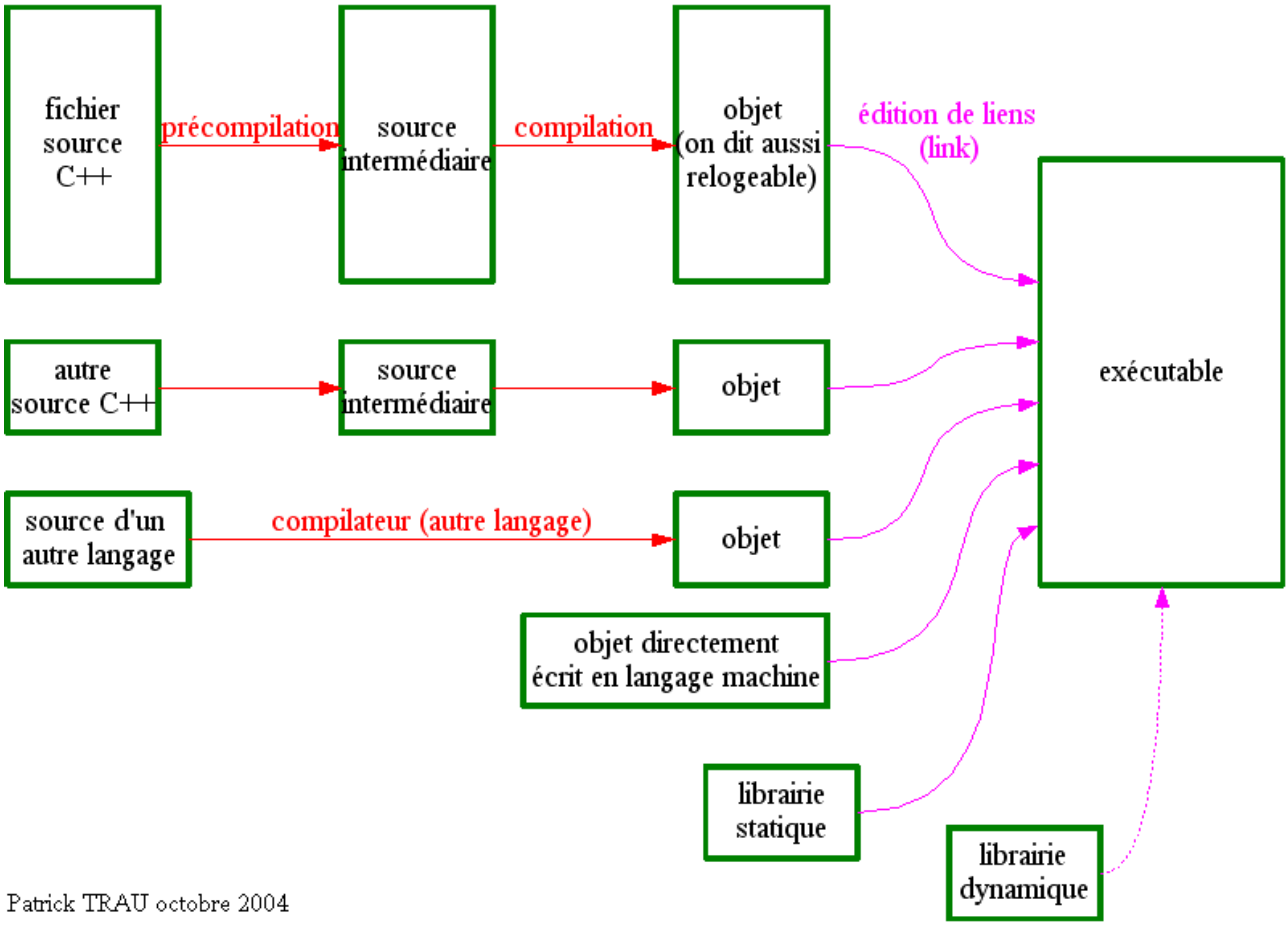
LES LANGAGES SEMI-COMPILÉS

Entre les deux, certains langages utilisent un **code intermédiaire** (ou P-code) : le code source est en partie compilé et le fichier intermédiaire est ensuite interprété mais avec une meilleure performance.
Par exemple dans le cas du langage Java, le code intermédiaire peut être comparé à du code binaire pour un processeur virtuel qui sera émulé par la Java-VM lors de l'exécution. La portabilité d'une application Java d'un machine à une autre ne dépend alors plus que de la Java-VM qui doit être adaptée à la machine destination, ce qui est nettement moins complexe que la réécriture d'un compilateur complet.

Parfois le source interprété n'est pas exécuté ligne à ligne mais par bloc (voire entièrement compilé au vol). C'est le cas des langages comme PERL ou Python.
Les temps exécution de ces langages interprétés se rapprochent (sans jamais les égaler) de ceux des langages compilés.

Exemple de construction d'un exécutable compilé et linké selon ses différentes étapes : depuis le fichier source jusqu'au binaire exécutable.

Le processus de compilation



E - HTML et Javascript

HTML (Hyper Text Markup Langage) est une application créée à l'origine pour les besoins en documentation du CERN (Centre Européen de Recherche Nucléaire) par Tim Berners Lee.
Il s'agit d'un langage de balisage. Les balises commencent par '<' et se terminent par '>'.

Il a eu plusieurs versions. La version actuelle (HTML 5) a été augmentée pour standardiser le multimédia et la disparité des formats d'écran (ordinateurs, phablettes, smartphones etc.)

Javascript (JS) est un langage (indépendant bien qu'inspiré de Java) qui permet l'exécution d'algorithme par un navigateur (javascript-client : Client Side JavaScript ou CSJS) ou par un site web (javascript-serveur : Serve Side JavaScript ou SSJS).
Créé à l'origine par Netscape ®, ne demeure que le javascript-client, le pendant serveur ayant été remplacé par nombre de langages interprétés comme le PHP puis plus récemment par Node-js.
Les récentes évolutions (Ajax, etc.) permettent d'accélérer l'interface web des navigateurs en autorisant JS à modifier une partie des pages web affichées sans réafficher toute la page.

De nombreux outils en ligne permettent de se former à HTML, aux feuilles de style CSS et aux rudiments de JavaScript.
C'est le cas en particulier du site du Consortium W3C (World Wide Web Consortium), organisme qui standardise le HTML : https://www.w3.org/ et de son école https://www.w3schools.com/ .

F - Annexe : références

RÉFÉRENCES

- Codages et convertisseurs

https://fr.wikipedia.org/wiki/American_Standard_Code_for_Information_Interchange

http://www.asciitable.com/

https://fr.wikipedia.org/wiki/Table_des_caract%C3%A8res_Unicode_(0000-0FFF)

https://en.wikipedia.org/wiki/UTF-8

https://en.wikipedia.org/wiki/Unicode

https://fr.wikipedia.org/wiki/IEEE_754

http://hapax.qc.ca/conversion.fr.html

https://fr.wikipedia.org/wiki/Portable_Network_Graphics

- Algorithmes

https://fr.wikipedia.org/wiki/Donald_Knuth
https://fr.wikipedia.org/wiki/Algorithmique

https://fr.wikipedia.org/wiki/Analyse_de_la_complexit%C3%A9_des_algorithmes
https://fr.wikipedia.org/wiki/Th%C3%A9orie_de_la_complexit%C3%A9_(informatique_th%C3%A9orique)

- HTML, CSS et JavaScript

https://www.w3.org/html/
https://www.w3schools.com/

TABLES

Les 128 caractères de la table ASCII originale.

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Le principe du codage UTF

Bits of code point	First code point	Last code point	Bytes in sequence	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
7	U+0000	U+007F	1	0xxxxxxx					
11	U+0080	U+07FF	2	110xxxxx	10xxxxxx				
16	U+0800	U+FFFF	3	1110xxxx	10xxxxxx	10xxxxxx			
21	U+10000	U+1FFFFF	4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx		
26	U+200000	U+3FFFFFF	5	111110xx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	
31	U+4000000	U+7FFFFFFF	6	1111110x	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx

Exemple de codage UTF pour 4 caractères

Character		Binary code point	Binary UTF-8	Hexadecimal UTF-8
$	U+0024	0100100	00100100	24
¢	U+00A2	00010100010	11000010 10100010	C2 A2
€	U+20AC	0010000010101100	11100010 10000010 10101100	E2 82 AC
𐍈	U+10348	00001000001101001000	11110000 10010000 10001101 10001000	F0 90 8D 88