# Représentation des Chaînes de Caractères en Informatique

Dans le domaine de la programmation, les chaînes de caractères permettent de manipuler du texte, qui est une forme de données très commune. Mais comment sont-elles représentées en informatique ? C'est une question importante pour comprendre le fonctionnement interne des ordinateurs et des langages de programmation.

5TTR 6TTR Exploration

---

# Représentation Binaire

Tout d'abord, il est crucial de se rappeller et de comprendre que les ordinateurs ne traitent que des données binaires, c'est-à-dire des suites de 0 et de 1. Chaque caractère dans une chaîne de caractères est donc représenté en interne par une série de bits (0 et 1).

# Encodage

Pour passer d'un caractère à sa représention binaire, il est nécessaire de passer par **une norme de codage**. C'est cette norme de codage qui va déterminer comment... coder un caractère sous forme binaire.

## ASCII

Au départ, aux Etats-Unis, il avait été décidé qu'**1 caractère serait encodé sur 7 bits**, ce qui permettait de représenter 2^7 caractères différents, soit **127 caractères possibles**. C'était largement suffisant pour pouvoir encoder les lettres de l'alphabet (majuscules et minuscules), les chiffres, les signes de ponctuation et quelques caractères spéciaux ou invisibles (tels que les retours à la ligne et les tabulations). C'est ce qu'on appelle la norme **ASCII**.

Cette norme **associe un caractère à un numéro dans une table**. Par exemple, en ASCII, la lettre `A` porte le numéro 65 et est donc codée en binaire comme `1000001` (note: il y a bien 7 bit).

La table ci-dessous montre toutes les valeurs possibles pour l'ASCII. La colonne "Char" montre les caractères correspondants.

# 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] |

(les caractères spéciaux invisibles ou non imprimables sont affichés *[entre crochets.]*)

Il ne faut bien évidemment pas connaître ces codes par coeurs...

## ASCII étendu

Ensuite, avec le développement de l'infomatique en Europe, il a fallu aussi supporter les caractères accentués (é, ù, ï...) et 127 caractères ne suffisaient plus. On a alors ajouté 1 bit pour l'encodage des caractères, soit 8 bit au total et, donc, **1 octet**.

Il est alors devenu possible d'encoder **255 caractères** différents. Largement de quoi supporter les accents et toute une nouvelle série de caractère spéciaux. C'est ce qu'on a appelé l'**ASCII *étendu***.

L'ASCII étendu est compatible avec l'ASCII: en effet, on a gardé les mêmes codes pour les caractères communs, et les nouveaux caractères ont été ajoutés à la suite des anciens.

Par exemple, en ASCII étendu, le caractère `A` vaut toujours `65` et le 'À' (avec accent) vaut `192`.

# Unicode et UTF-8

Ensuite, les informaticiens ont voulu avoir la possibilité de **représenter de manière unifiée *tous* les systèmes d'écriture du monde**, offrant ainsi une solution globale pour le codage des caractères textuels. Autant dire, qu'avec 255 caractères, c'était impossible.

C'est ainsi que l'**Unicode** et l'**UTF-8** sont nés.

**Unicode** attribue un numéro unique à chaque caractère de tous les systèmes d'écriture du monde. Le répertoire Unicode peut contenir plus d'un million de caractères, ce qui est effectivement bien trop grand pour être codé par un seul octet

Unicode est rétro-compatible avec l'ASCII. Autrement dit, les 127 premiers caractères Unicode correspondent aux 127 caractères de l'ASCII.

> 💡 Tous les émojis ont été encodés dans Unicode... les émojis sont donc des caractères à part entière.
> 😁 Tu peux d'ailleurs facilement intégrer des emojis à tes textes dans Word, dans du HTML... 🤡 🦄 🍕 🌞

> Avec Windows, tu peux facilement insérer un émoji dans ton texte à l'aide du raccourci-clavier `WIN-;` (voir la section 'Liens')

**UTF-8**, quant à elle, est une **norme de codage**, qui détermine la façon dont les caractères Unicode vont être **stockés sous forme de bit**. Elle utilise une longueur de codage variable (1 à 4 octets) pour représenter les caractères Unicode. Par exemple, la lettre `À` est codée sur un octet, tandis que le signe € est codé sur 3 octets.

> Essaie toi-même: crée un fichier .txt sur ton disque qui contiendra un seul carcatère. Enregistre et vérifie la taille du fichier dans l'explorateur de fichiers. Que constates-tu? Essaie avec différents caractères (é, È, ~, €...).

En 2020, UTF-8 était utilisé par plus de 95% des sites web dans le monde.

# Exemple Concret

Imaginons une chaîne de caractères simple : "Bonjour".

En ASCII, chaque lettre est représentée par un code à 7 bits. Par exemple, le "B" majuscule est 66 en ASCII, ce qui correspond à 1000010 en binaire. Le mot "Bonjour" sera donc encodé, en ASCII, en binaire: `1000010 1101111 1101110 1101010 1101111 1110101 1110010`.

En UTF-8, "Bonjour" serait codé différemment, surtout si des caractères spéciaux ou des lettres accentuées étaient présents.

# Conclusion

Comprendre la représentation des chaînes de caractères est fondamental pour la programmation et le traitement de données. Cela influence la manière dont les programmes interagissent avec les données textuelles et la façon dont ces données sont stockées et transmises. En tant que professeur d'informatique, il est important de familiariser les élèves avec ces concepts pour leur permettre de développer une compréhension approfondie du fonctionnement des systèmes informatiques.

Vidéo à la une à regarder sur Youtube: http://youtu.be/uVNyQZQY2mY