Ce qu'il faut comprendre

Rapide résumer

Un ordinateur est un outil qui effectue des calculs de l'information. Les informations sont représentées en binaire. Pour programmer un ordinateur, on utilise un langage de programmation et des algorithmes. Un algorithme est une liste d'instructions pour répondre à une problématique. On peut analyser les performances mémoires et d'exécution d'un algorithme grâce à la complexité.

L'encodage

Le binaire

Le binaire représente l'information grâce à des bits. Un bit est un état qui peut être soit vrai soit faux (1 ou 0). Pour représenter un booléen, qui est une valeur qui peut être soit vraie soit fausse, on utilise un seul bit. Pour représenter un nombre entier, on utilise plusieurs bits que l'on décompose en puissances de deux, ainsi qu'un bit qui indique le signe :

Nombre	Signe	$2^6 = 64$	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
24	0	0	0	1	1	0	0	0
-36	1	0	1	0	0	1	0	0

Les nombres décimaux sont plus difficiles à représenter car les puissances de deux ne permettent pas de décomposer les parties décimales sans recourir à un trop grand nombre de bits. Il existe trois formats pour représenter les nombres décimaux :

Format	Avantages	Inconvénients	Utilisation
Double : place la virgule au milieu des bits.	Facile à manipuler et précision fixe.	Taille limitée pour la partie entière. Nécessite plus de place en mémoire.	Lorsque l'on a besoin de précision pour les calculs décimaux.
Flottant (norme IEEE754) : La virgule se déplace en fonction de la valeur.	Permet de représenter de plus grandes valeurs entières en faisant varier la précision. Prends moins de place en mémoire que les autres options.	La précision n'est pas fixe. La précision n'est donc pas garantie.	Lorsque la précision n'est pas un élément critique.
Décimal : Représente les nombres comme du texte au format décimal.	Très grande précision.	Prends beaucoup de place en mémoire. Les calculs et les opérations sont moins efficaces.	Secteurs spécifiques comme les banques ou le médical.

Pour représenter les textes on utilise les nombres et une table d'encodage, comme la table ASCII ou la table UTF.

Pour représenter une séquence, on juxtapose les valeurs les unes à la suite des autres dans la mémoire.

L'hexadécimal

L'hexadécimal représente les nombres en utilisant 16 chiffres, de 0 à F. Ce format est souvent pour représenter des couleurs. Pour distinguer entre une valeur binaire et une valeur hexadécimale, on utilise les préfixes '0b' et '0x'.

Portes binaires

Les portes binaires sont des opérations sur les bits. Pour visualiser les portes, on utilise les tables de vérités, les symboles arithmétiques ou les diagrammes électroniques. Il y en a 4 couramment utilisées :

- La porte **NON**, notée '¬', retourne la valeur opposée du bit.
- La porte **ET**, notée '.', retourne vraie si et seulement si les deux bits sont vrais.
- La porte **OU**, notée '+', retourne vraie si au moins l'un des deux bits est vrai.

- La porte **XOU**, notée '⊕', retourne vraie si seulement l'un des deux bits est vrai.

Algèbre de Boole : propriétés

- Priorités :

Les langages de programmation

Pour programmer un ordinateur, on utilise des instructions machines. Ce sont des séries d'octets indiquant à l'ordinateur quelles actions il doit exécuter et avec quelle(s) valeur(s). Pour simplifier le développement, on utilise un langage de programmation.

Il existe différents langages de programmations plus ou moins pertinents en fonction du contexte. Il y a des langages conçus pour des domaines spécifiques (PHP, Swift, OCaml, …), des langages plus performants que d'autres (C, Rust, C++, …) et des langages plus ergonomiques (Python, Javascript, Ruby, …).

L'algorithmie

Un algorithme est une suite d'instructions permettant de résoudre un problème. Un algorithme doit être clair, utiliser un vocabulaire précis et être compréhensible par tous.

Types de données

Les types atomiques :

- **BOOL** : Booléen
- **INT / UINT** : Entier
- **CHAR** : caractère
- **STRING** : TEXTE

- **FLOAT / DOUBLE / DECIMAL** : Valeur décimale

Les types composés :

- **STRUCTURE** : Regroupe plusieurs valeurs sous un même nom.
- **ENUM** : Liste plusieurs options.
- **UNION** : Indique qu'une valeur peut être de différents types.

Les collections :

- **ARRAY** : une séquence de taille fixe de valeurs de même type. Les éléments sont accessibles via leurs index.
- **TUPLE** : Tableau immutable qui peut contenir des valeurs de différents types.

Structures de données

Une structure de données permet d'organiser des informations. Elles permettent de représenter des relations, des priorités, etc…

- **LIST : liste chainée (ou doublement chainée) :** Fonctionne comme un tableau mais la taille n'est pas fixe.
- **STACK : pile :** Fonctionne comme une liste mais les éléments ne sont pas accessibles via leurs index. Le premier élément ajouté dans la pile est le dernier à sortir (FILO : First In Last Out).
- **QUEUE : file** : Fonctionne comme une liste mais les éléments ne sont pas accessibles via leurs index. Le premier élément ajouté dans la file est le premier à sortir (FIFO : First In First Out).
- **MAP : dictionnaire** : Associe des clés à des valeurs. Il ne peut pas y avoir plusieurs fois la même clé, mais il peut y avoir plusieurs fois la même valeur.
- **GRAPH : graphes** : Représente des relations entre des objets (GPS, réseaux social, …).
- **TREE : arbre** : Il s'agit d'une structure hiérarchique de parent à enfants qui représente des relations entre des objets.

Ecrire un algorithme

- Les variables sont nommées en respectant la convention « snake case » ou « camel case ».
- Les fonctions et procédures sont nommées en respectant la convention « snake case » ou « camel case ».
- Les structures, énumérations et unions sont nommées en respectant la convention « snake case » ou « pascal case ».
- Une fonction retourne une valeur alors qu'une procédure ne renvoie rien.

Utilisation d'une variable :

```
INT number
number = 10
print(number)
INT values[100]
print("First value :", values[0])
print("Last value : ", values[values.length])
```

Opérations binaires et arithmétiques et comparaisons :

- '+' pour l'addition et la concaténation de texte.
- '++' pour l'incrément d'une variable de 1.
- '+=' pour l'incrément d'une variable'.
- '-' pour la soustraction.
- '--' pour l'incrément d'une variable de 1.
- '-=' pour l'incrément d'une variable.
- '*' pour la multiplication.
- '/' pour la division.
- '%' pour le reste de la division euclidienne.
- ' !' pour la porte binaire NON.
- '&' pour la porte binaire ET.
- '&&' pour la porte binaire « raccourcie » ET.
- '|' pour la porte binaire OU.
- '||' pour la porte binaire « raccourcie » OU.
- '^' pour la porte binaire XOU.
- '<<' pour effectuer un décalage de bits à gauche.
- '>>' pour effectuer un décalage de bits à droite.
- '<' pour vérifier si une valeur est plus petite qu'une autre.

- '<=' pour vérifier si une valeur est plus petite ou égale à une autre.
- '>' pour vérifier si une valeur est plus grande qu'une autre.
- '>=' pour vérifier si une valeur est plus grande ou égale à une autre.
- '=' pour assigner une valeur.
- '==' pour vérifier si deux valeurs sont égales.
- ' !=' pour vérifier si deux valeurs ne sont pas égales.

Conditions :

```
IF (condition) THEN
    Actions …
ELIF (condition 2) THEN
    Actions …
ELSE
    Actions …
ENDIF
```

```
SWITCH (valeur) THEN
     CASE (valeur 1) THEN
          Actions …
     BREAK
     CASE (valeur 2) THEN
          Actions …
     BREAK
     DEFAULT THEN
          Actions …
     BREAK
ENDSWITCH
```

Boucles :

```
LOOP THEN
      Actions …
ENDLOOP
```

```
WHILE (condition) THEN
      Actions …
ENDWHILE
```

```
DO
    Actions …
WHILE (condition)
```

```
FOR (INT i = 0; i < 10; i++) THEN
    Actions …
ENDFOR
```

Les fonctions et procédures :

```
FONCTION (TYPE1 paramètre1, TYPE2 paramètre2, …) TYPE THEN
    Actions …
    RETURN valeur
ENDFONCTION
```

Structures, énumérations et unions :

```
STRUCTURE nom
    TYPE1 attribut1
    TYPE2 attribut2
    …
ENDSTRUCTURE
```

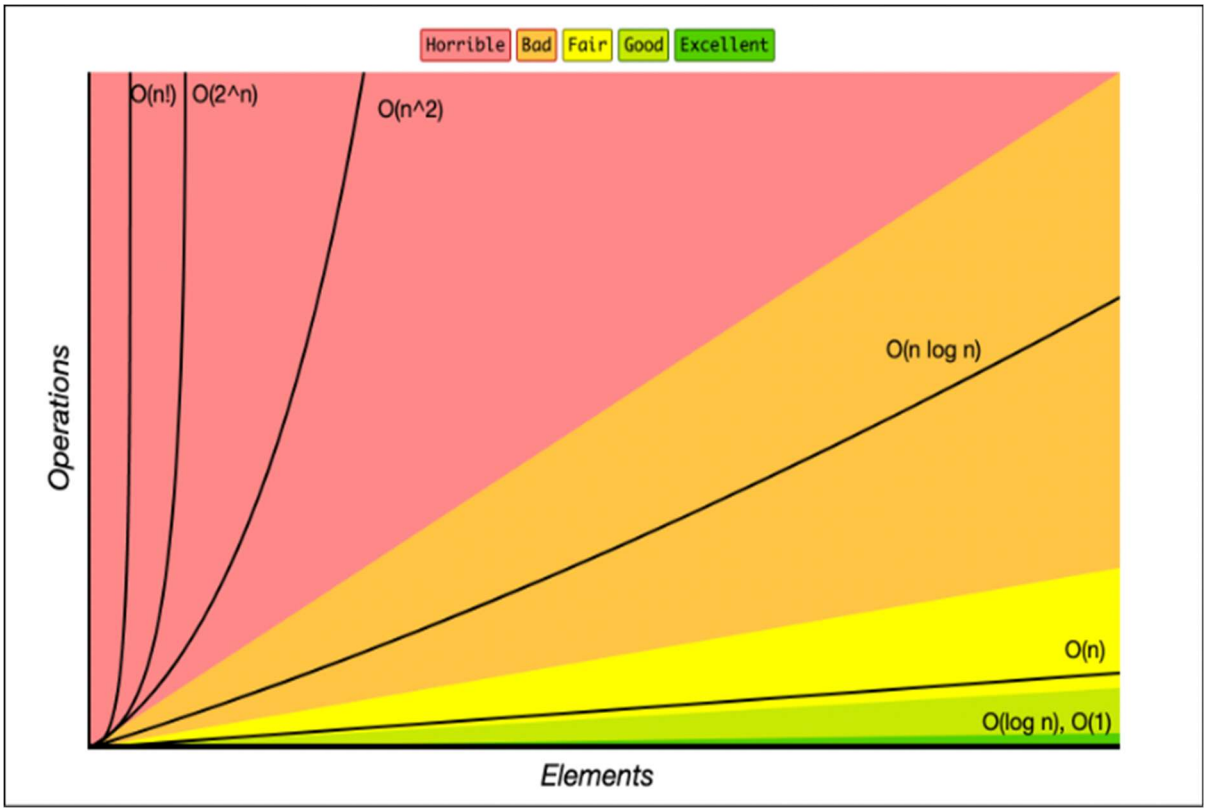
```
ENUM nom
    Option1
    Option 2
    …
ENDENUM
```

```
UNION nom
    TYPE1 variante1
    TYPE2 variante2
    …
```

Complexité

La complexité permet d'évaluer le temps d'exécution et l'empreinte mémoire d'un algorithme. On utilise la notation « Big O » qui évalue par ordre de grandeur la complexité d'un algorithme. Cette notation évalue en fonction de 'n', l'évolution de l'empreinte mémoire et du temps d'exécution. 'n' peut correspondre à la taille d'une collection, d'un nombre d'itérations, etc… La notation « Big O » utilise principalement ces valeurs remarquables :

- $O(1)$
- $O(\log n)$
- $O(n)$
- $O(n \log n)$
- $O(n^2)$
- $O(2^n)$
- $O(n!)$



Source : https://www.freecodecamp.org/news/big-o-cheat-sheet-time-complexity-chart/