Les tableaux de bits

Vous avez déjà rencontré la classe `std::bitset` dans le chapitre [Logique binaire et calcul booléen](#), pour afficher une séquence de bits.

main.cpp

```
#include <iostream>
#include <bitset>

int main() {
    std::cout << "0b" << std::bitset<8>(0b101010) << std::
endl;
    std::cout << "0b" << std::bitset<8>(42) << std::endl;
}
```

affiche :

```
0b00101010
0b00101010
```

Ce chapitre détaille l'utilisation de cette classe `std::bitset` et les notions de *flag* et *mask*. Les notions vues dans les chapitres [Logique binaire et calcul booléen](#) et [\[Aller plus loin\] L'algèbre de Boole](#) seront utilisées, n'hésitez pas à les relire si nécessaire.

Représentation binaire

La notion de représentation binaire a été vue pour les entiers et les nombres réels. Pour rappel, la représentation binaire d'une valeur est une séquence de bits telle qu'elle est conservée en mémoire et manipulée par les processeurs.

La représentation binaire n'est généralement pas manipulée directement par le développeur, sauf besoin particulier. La raison est qu'une

représentation binaire ne signifie rien en elle-même. Elle n'a de sens que lorsqu'elle est associée à un type. Le type est donc la façon dont le processeur doit interpréter une séquence de bits. Dit autrement, une même séquence de bits peut correspondre à différentes valeurs, selon le type.

main.cpp

```
#include <iostream>

int main() {
    const int i { 0b00110001 };
    std::cout << i << std::endl;
    const char c { 0b01100001 };
    std::cout << c << std::endl;
}
```

affiche :

```
49
a
```

L'interprétation d'une séquence de bits en valeur va dépendre d'une convention : les normes d'encodage. Il en existe plusieurs selon le type et parfois plusieurs pour un même type (vous avez par exemple peut être déjà entendu parler de "ASCII" ou de "UTF-8", qui sont des normes de codage des caractères). Mais certaines normes se sont généralement imposées pour les cas d'utilisation les plus courants.

Il est possible de citer par exemple :

- le [complément à 2](#) pour les nombres entiers relatifs (nombres signés) ;
- la norme [IEEE 754](#) pour les nombres réels ;
- la norme [ASCII](#) pour les caractères de base en informatique ;
- les normes CP850, ISO 8859-1, UTF, et [bien d'autres encore](#), pour les caractères.

(Les caractères sont un cas particulier, du fait de la multitude d'alphabets existants dans le monde, ce qui explique qu'il existe plusieurs normes.

Cela sera détaillé dans le chapitre sur l'internationalisation).

Les nombres entiers non signés

bitset

créer un bitset

afficher un bitset

tester un bit : mask, flag, opérateur ET bit à bit, test() forcer un bit : OU bit a bit

tester plusieurs bit : count, all, any, none (cf algo)

Plusieurs fois des données de même type. Accès avec un indice, partant de 0. Taille fixé à la compilation ou à l'exécution : bitset à la compilation (vector<bool> à l'exécution).

Accès a un élément : [] Validation taille : assert connaitre la taille : size

Exercices

A partir de la représentation binaire d'un nombre (42 = 0b0000000000101010)

- Compter le nombre de 1 dans la représentation
- trouver la plus longue chaîne de 1 dans la représentation

Chapitre précédent	Sommaire principal	Chapitre suivant
------------------------------------	------------------------------------	----------------------------------