Comprendre l'opérateur (sizeof) en C++

La valeur **sizeof** d'une structure n'est pas toujours égale à la somme des valeurs **sizeof** de chaque membre. Cela est dû au remplissage ajouté par le compilateur pour éviter les problèmes d'alignement. Le remplissage n'est ajouté que lorsqu'un membre de la structure est suivi d'un membre de taille supérieure ou à la fin de la structure. Les contraintes d'alignement peuvent varier d'un compilateur à l'autre, car les normes C++ stipulent que l'alignement de la structure dépend entièrement de l'implémentation.

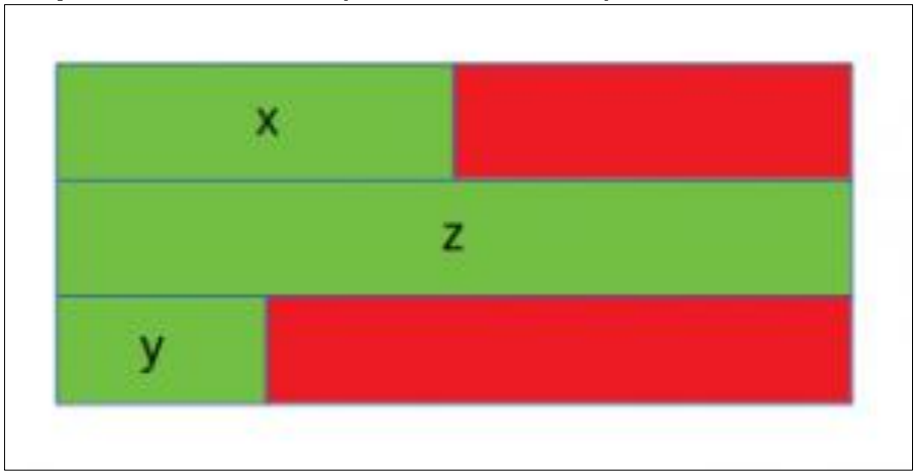
Considérons cette structure (main.cpp)

```cpp
struct A
{
    // sizeof(int) = 4
    int x;
    // Remplissage de 4 octets

    // sizeof(double) = 8
    double z;
    // Remplissage de 0 octets

    // sizeof(short int) = 2
    short int y;
    // Remplissage de 6 octets
};
```

Dans la théorie :
On a : (sizeof(int) = 4), (sizeof(double) = 8), (sizeof(short int) = 2)
Soit un total de : (4 + 8 + 2 = 14 octets)

Dans la pratique :
On a : (int(4) est suivi de double(8) : (8 > 4) : remplissage(4) : 8)
On a : (double(8) est suivi de short int(2) : (2 < 8) : remplissage(0) : 8)
On a : (short int(2) termine la structure)
On a : (short int(2) suit double(8) : (8 > 2) : remplissage(6) : 8)
Soit au total : (8 + 8 + 8 : **24 octets**).

Empreinte mémoire (Consommation)



La partie rouge représente le remplissage ajouté pour l'alignement des données, et la partie verte représente les membres de la structure. Dans ce cas, x (int) est suivi de z (double), dont la taille est supérieure à x . Le remplissage est donc ajouté après x . De plus, un remplissage est nécessaire à la fin pour l'alignement des données.

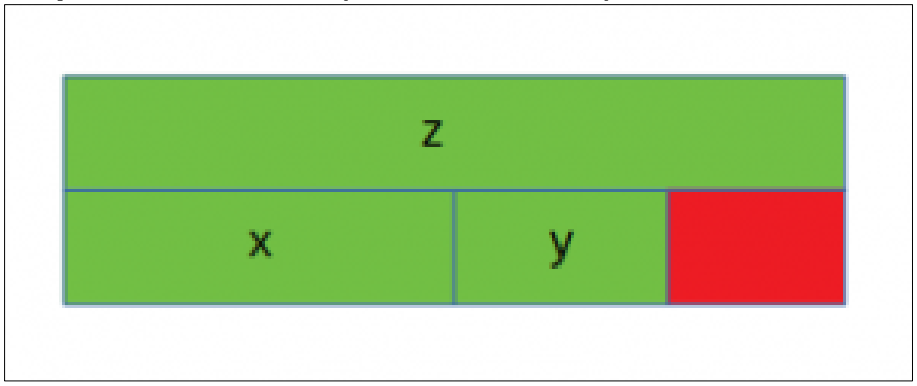
Résultat des tests (Terminal)

```
(1): Taille de la structure : 24 octets
```

Considérons cette structure (main.cpp)

```cpp
struct B
{
    // sizeof(double) = 8
    double z;
    // Remplissage de 0 octets

    // sizeof(int) = 4
    int x;
    // Remplissage de 0 octets

    // sizeof(short int) = 2
    short int y;
    // Remplissage de 2 octets
};
```

Empreinte mémoire (Consommation)



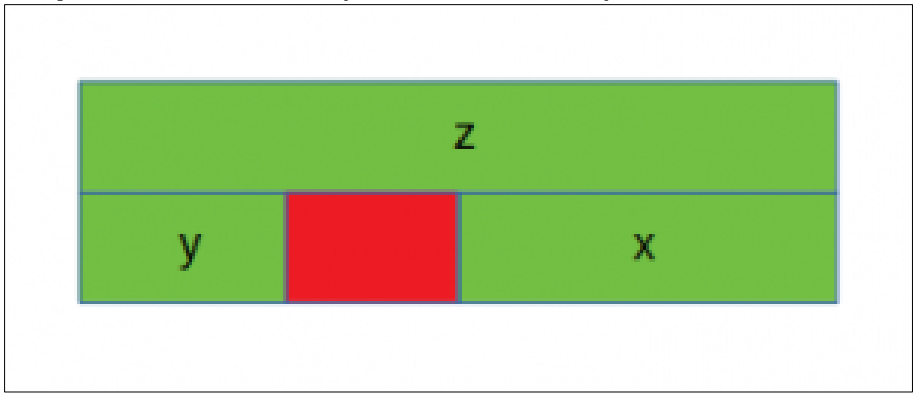
Résultat des tests (Terminal)

```
(2): Taille de la structure : 16 octets
```

Considérons cette structure (main.cpp)

```cpp
struct C
{
    // sizeof(double) = 8
    double z;
    // Remplissage de 0 octets

    // sizeof(short int) = 2
    short int y;
    // Remplissage de 2 octets

    // sizeof(int) = 4
    int x;
    // Remplissage de 0 octets
};
```

Empreinte mémoire (Consommation)



Résultat des tests (Terminal)

```
(3): Taille de la structure : 16 octets
```

Conclusion

Si vous souhaitez optimiser l'empreinte mémoire de vos structure de données dans vos projets C++, pour un gain en performance, il est recommandé de disposer les membres de chaque structure de telle manière à commencer leurs déclarations par les membres ayant les plus grandes tailles (sizeof) et de terminer par les membres ayant les plus petites tailles (sizeof).

Fin