Cours MOOC EPFL d'introduction à la programmation orientée objet, illustré en C++

Troisième devoir
Surcharge d'opérateurs

J.-C. Chappelier & J. Sam

1 Exercice 1 — Chimie

Un chimiste souhaite que vous l'aidiez à modéliser les *flacons* de produits chimiques qu'il manipule.

1.1 Description

Télécharger le programme `chimie.cc` fourni et le compléter suivant les instructions données ci-dessous.

ATTENTION : vous ne devez en aucun cas modifier ni le début ni la fin du programme fourni, juste ajouter vos propres lignes à l'endroit indiqué. Il est donc impératif de respecter la procédure suivante :

1. sauvegarder le fichier téléchargé sous le nom `chimie.cc` ou `chimie.cpp` ;
2. écrire le code à fournir (voir ci-dessous) entre ces deux commentaires :

```
/*****************************************
 * Complétez le programme à partir d'ici.
 *****************************************/

/*****************************************
 * Ne rien modifier après cette ligne.
 *****************************************/
```

3. sauvegarder et tester son programme pour être sûr(e) qu'il fonctionne correctement, par exemple avec les valeurs utilisées dans l'exemple de déroulement donné plus bas ;

4. soumettre le fichier modifié (toujours `chimie.cc` ou `chimie.cpp`) dans « My submission » puis « Create submission ».

Le code fourni :

— commence la déclaration de la classe `Flacon`, décrite plus bas ;
— puis définit une fonction utilitaire `afficher_melange` et ensuite, dans le `main()`, déclare plusieurs « flacons » et affiche leurs mélanges.

Le corps de la classe `Flacon` manque et c'est ce qu'il vous est demandé d'écrire.

Un « flacon » est caractérisé par :

— le *nom* du produit qu'il contient, comme par exemple « acide chlorhydrique » ;
— le *volume* (en ml) qu'il peut contenir, de type `double` ;
— et son *pH* (de type `double`).

La classe `Flacon` comportera de plus :

— un constructeur initialisant les attributs au moyen de valeurs passées en paramètre dans l'ordre illustré par le `main()` fourni ; il n'y aura pas de constructeur par défaut ;
— une méthode `etiquette`, de prototype
```
ostream& etiquette(ostream& sortie) const ;
```
affichant, dans le flot (`ostream`) passé en argument, l'étiquette à coller sur le « flacon ». Cette étiquette spécifiera le nom, le volume et le pH du contenu du « flacon » selon le format d'affichage suivant (d'autres exemples sont également présents dans l'exemple de déroulement fourni plus bas) :
```
Acide chlroydrique : 250 ml, pH 2
```
— une surcharge de l'opérateur d'affichage `<<` pour cette classe ; cette surcharge devra utiliser la méthode `etiquette` précédente.

Il devra enfin être également possible de mélanger deux « flacons » au moyen de l'opérateur + :
soient deux `Flacon`s de nom, volume et pH respectifs : `nom1`, `volume1`, `ph1` et `nom2`, `volume2`, `ph2` ; le mélange de ces deux *flacons* au moyen de l'opérateur + donne un nouveau « flacon » dont :

1. le nom est « *nom1* + *nom2* », par exemple : « Eau + Acide chlorhydrique » ;
2. le volume est la somme de `volume1` et `volume2` ;
3. et le pH est (que les chimistes nous pardonnent cette grossière approximation !) :

$$\mathtt{pH} = -\log_{10}\left(\frac{\mathtt{volume1}\times 10^{-\mathtt{ph1}} + \mathtt{volume2}\times 10^{-\mathtt{ph2}}}{\mathtt{volume1}+\mathtt{volume2}}\right)$$

La fonction $\log_{10}$ s'écrit `log10` en C++ et pour faire le calcul 10^{-x} utiliser l'expression `pow(10.0, -x)`.

Un exemple de déroulement est fourni plus bas. Sur cet exercice, nous donnerons un bonus de 5 points à tous ceux qui munissent également la classe `Flacon` de l'opérateur interne `operator+=` permettant de faire le mélange dans le *flacon* lui-même : même si cela n'est pas très réaliste du point de vue physique, un tel « flacon » (d'où les guillemets) verra son volume augmenter comme décrit ci-dessus ; en clair : après `a += b;` le volume `a` est effectivement modifié (ainsi que son nom et son pH).

Si vous le faites et voulez prétendre au bonus (noté sur 35 points), ajoutez simplement la ligne suivante dans votre fichier :

```
#define BONUS
```

(sinon vous serez simplement notés sur 30 points).

1.2 Exemples de déroulement

```
Si je mélange
"Eau : 600 ml, pH 7"
avec
"Acide chlorhydrique : 500 ml, pH 2"
j'obtiens :
"Eau + Acide chlorhydrique : 1100 ml, pH 2.34242"
Si je mélange
"Acide chlorhydrique : 500 ml, pH 2"
avec
"Acide perchlorique : 800 ml, pH 1.5"
j'obtiens :
"Acide chlorhydrique + Acide perchlorique : 1300 ml, pH 1.63253"
```

2 Exercice 2 — Construction

Le but de cet exercice est d'écrire un programme permettant de décrire des « constructions » à base de « briques de base » en utilisant les opérateurs du langage C++.

Cet exercice est plus simple qu'il n'y parrait. Ne cherchez pas *trop* compliqué et suivez la méthodologie proposée (section 2.2).

2.1 Description

Télécharger le programme `construction.cc` fourni et le compléter suivant les instructions données ci-dessous.

ATTENTION : vous ne devez en aucun cas modifier ni le début ni la fin du programme fourni, juste ajouter vos propres lignes à l'endroit indiqué. Il est donc impératif de respecter la procédure suivante :

1. sauvegarder le fichier téléchargé sous le nom `construction.cc` ou `construction.cpp`;
2. écrire le code à fournir (voir ci-dessous) entre ces deux commentaires :

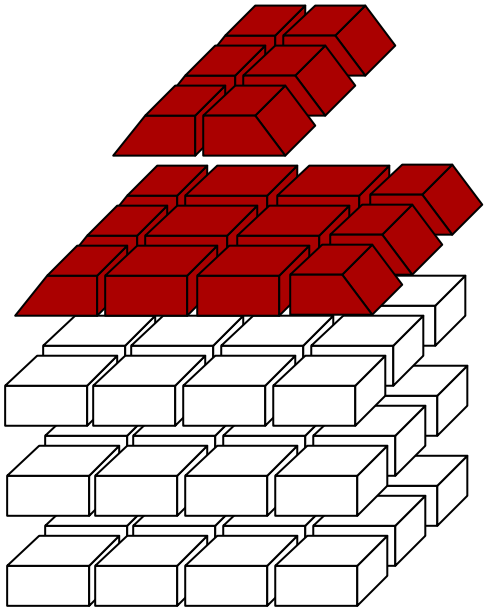
```
/*****************************************
 * Complétez le programme à partir d'ici.
 *****************************************/

/*****************************************
 * Ne rien modifier après cette ligne.
 *****************************************/
```

3. sauvegarder et tester son programme pour être sûr(e) qu'il fonctionne correctement, par exemple avec les valeurs utilisées dans l'exemple de déroulement donné plus bas ;
4. soumettre le fichier modifié (toujours `construction.cc` ou `construction.cpp`) dans « My submission » puis « Create submission ».

Le code fourni

— déclare des types `Forme` et `Couleur` qui nous seront utiles pour les « briques de base » ; pour simplifier, nous avons ici choisi simplement des `string` ;
— commence la déclaration de la classe `Brique`, décrite plus bas et servant à représenter les « briques de base » ;
— puis dans le `main()`, déclare plusieurs « briques de base » et construit une « maison » que l'on pourrait représenter graphiquement comme ceci :

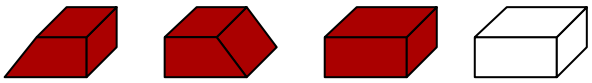


Un exemple de déroulement possible de ce code est fourni plus bas.

La fin de la définition de la classe `Brique`, et surtout la définition de la classe `Construction`, décrites ci-dessous, manquent et il vous est demandé de les fournir.

2.1.1 La classe `Brique`

La classe `Brique` sert à représenter les « briques de base » de nos constructions, que l'on pourrait illustrer comme ceci :



Chaque `Brique` est caractérisée par une forme et une couleur, telles que déjà données dans le fichier `construction.cc` fourni.

Nous vous demandons de terminer la définition de la classe `Brique` fournie en :

- lui ajoutant un constructeur prenant deux paramètres, respectivement de type `Forme` et `Couleur` (dans cet ordre) ; il n'y aura *pas* de constructeur par défaut pour cette classe ;
- lui ajoutant une méthode publique de prototype
  ```
  ostream& afficher(ostream& sortie) const ;
  ```
  lui permettant d'afficher son contenu au format :
  - si la couleur n'est pas la chaîne vide :
    ```
    (forme, couleur)
    ```
    comme par exemple (d'autres exemples sont fournis plus bas) :

`(obliqueG, rouge)`

- si la couleur est la chaîne vide, afficher simplement la forme (sans autre signe) ;

— surchargeant l'opérateur d'affichage << pour cette classe ; cette surcharge devra utiliser la méthode `afficher` précédente.

2.1.2 Contenu de la classe `Construction`

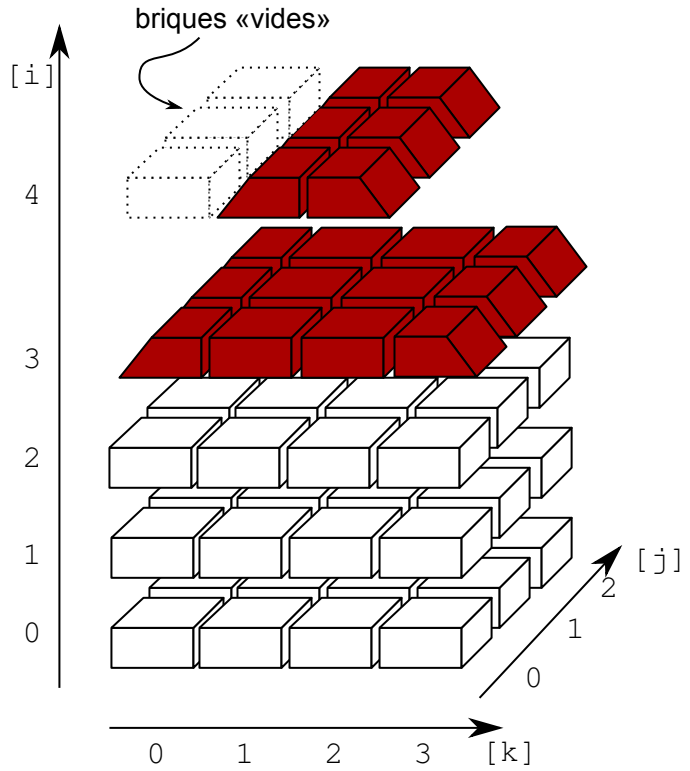
La classe `Construction` nous permettra de représenter les « constructions » dans notre programme. Une « construction » est simplement un ensemble de briques en 3 dimensions.

Cette classe devra donc avoir un attribut `contenu` (respecter strictement ce nom) qui est un tableau dynamique à 3 dimensions de « briques de bases », i.e. un `vector` de `vector` de `vector` de `Brique`.

Pour les besoins de la correction, **IL EST IMPERATIF DE FAIRE COMMENCER VOTRE CLASSE `Construction` DE LA FAÇON SUIVANTE :**

```
class Construction
{
 friend class Grader;
```

L'ordre de représentation interne des 3 dimensions (dans quel sens vous voyez la construction) sera le suivant :



Cela veut dire que le premier indice (i ci-dessus) représente la hauteur de la construction, le second (j) la profondeur et le dernier (k) la largeur.

Ajouter ensuite à la classe `Construction` :

— un constructeur prenant une `Brique` comme paramètre et construisant le `contenu` comme un tableau 1x1x1 contenant cette seule brique ;

— une méthode publique de prototype

```
ostream& afficher(ostream& sortie) const ;
```

affichant son contenu couche par couche comme illustré plus bas dans l'exemple de déroulement (un message « `Couche` *`numéro`* `:` » est intercalé entre l'affichage de chaque couche de la construction) et n'affichant rien si le contenu de la construction est vide.

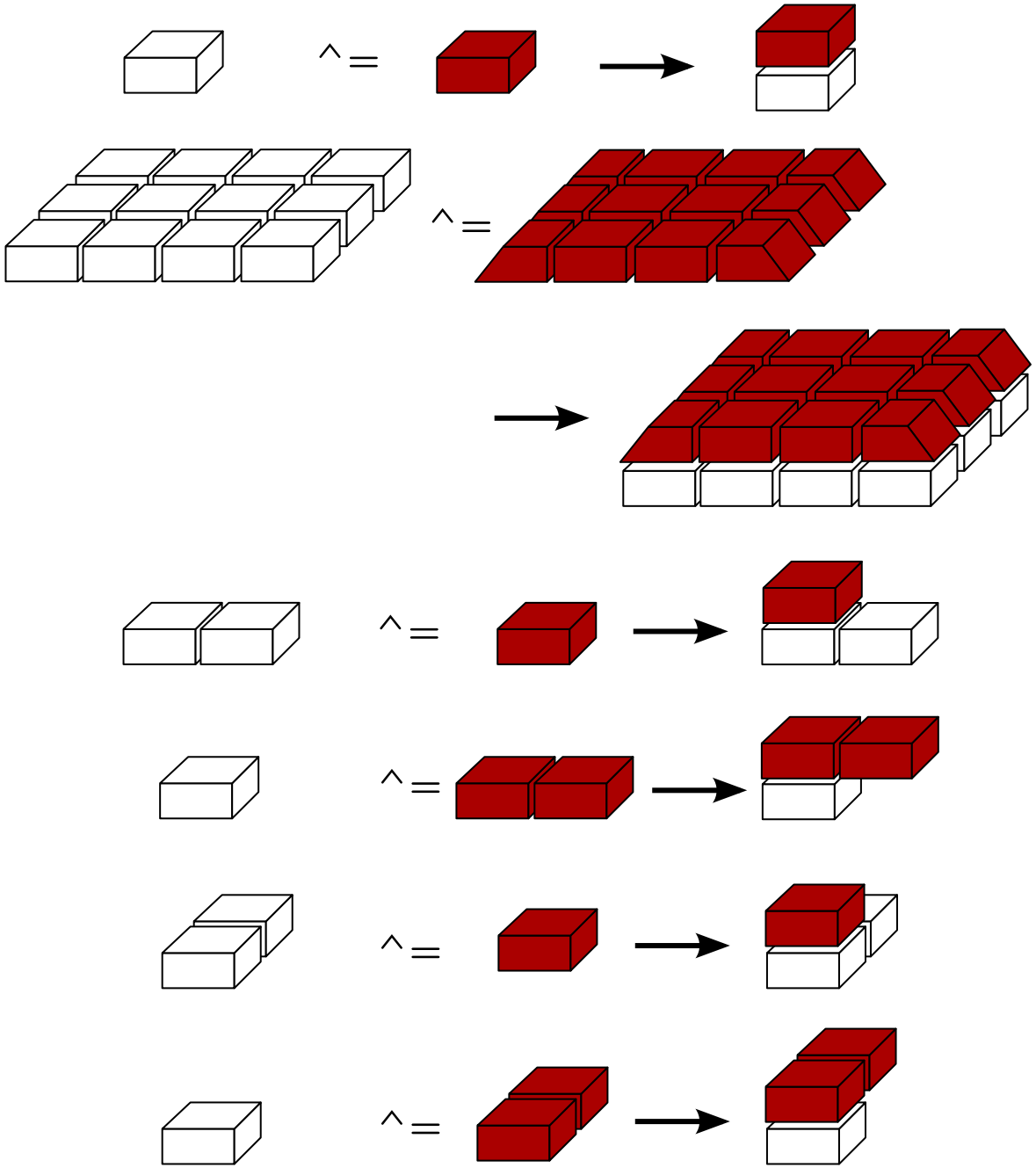
2.1.3 Opérateurs pour de la classe `Construction`

Pour pouvoir afficher et surtout décrire de façon compacte les constructions, nous vous demandons d'ajouter à la classe `Construction` les dix opérateurs sui-

vants [1] :

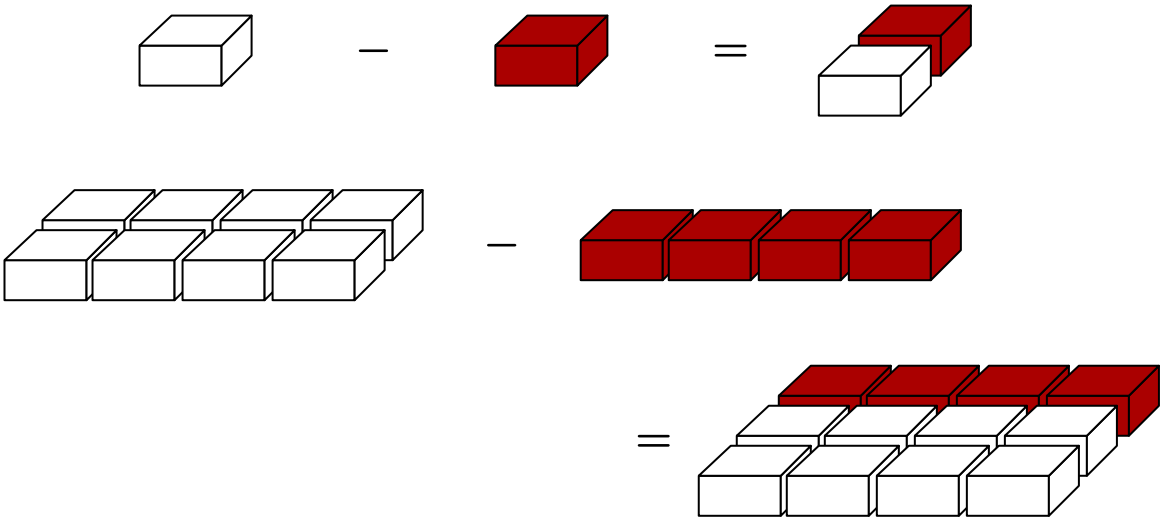
1. l'opérateur (externe) d'affichage << utilisant la méthode `afficher` précédente ;
2. l'opérateur interne `operator^=` et l'opérateur externe `operator^` qui ajoutent une `Construction` au dessus :
   — `a ^= b;` ajoute la `Construction b` au dessus de la `Construction a`, comme illustré ici :

1. Ces opérateurs servent simplement à décrire les couches de la constructions. Ils ne se préoccupent pas de la réalité physique, ou non, du résultat.



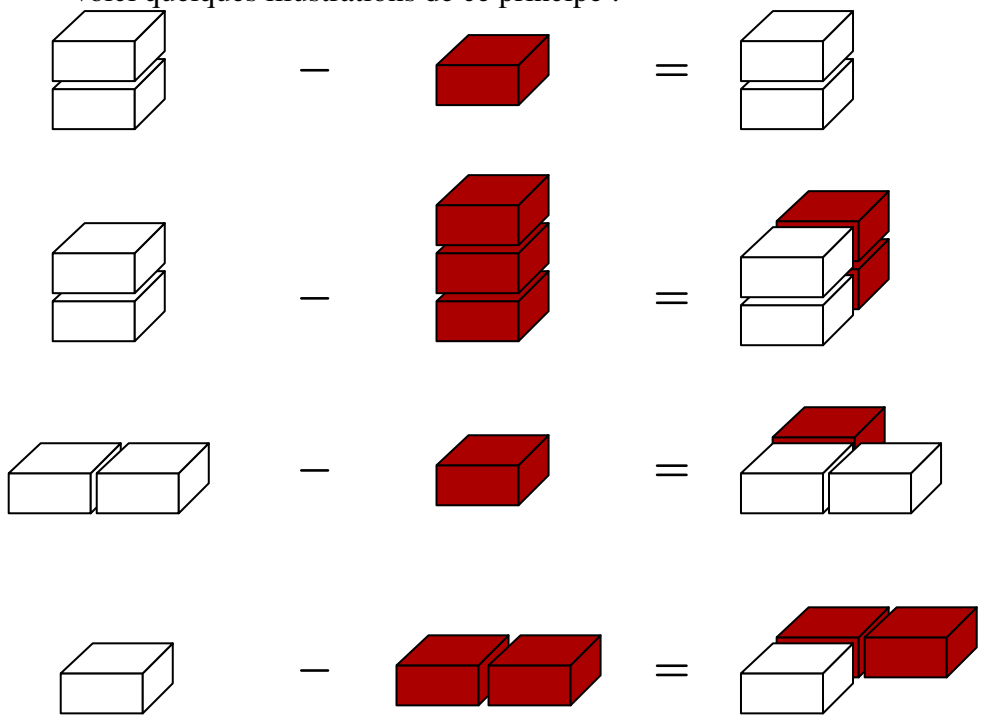
— `a ^ b;` crée une nouvelle `Construction` qui est le résultat de la `Construction b` mise au dessus de la `Construction a` ;

3. l'opérateur interne `operator-=` et l'opérateur externe `operator-` qui ajoutent une `Construction` derrière :

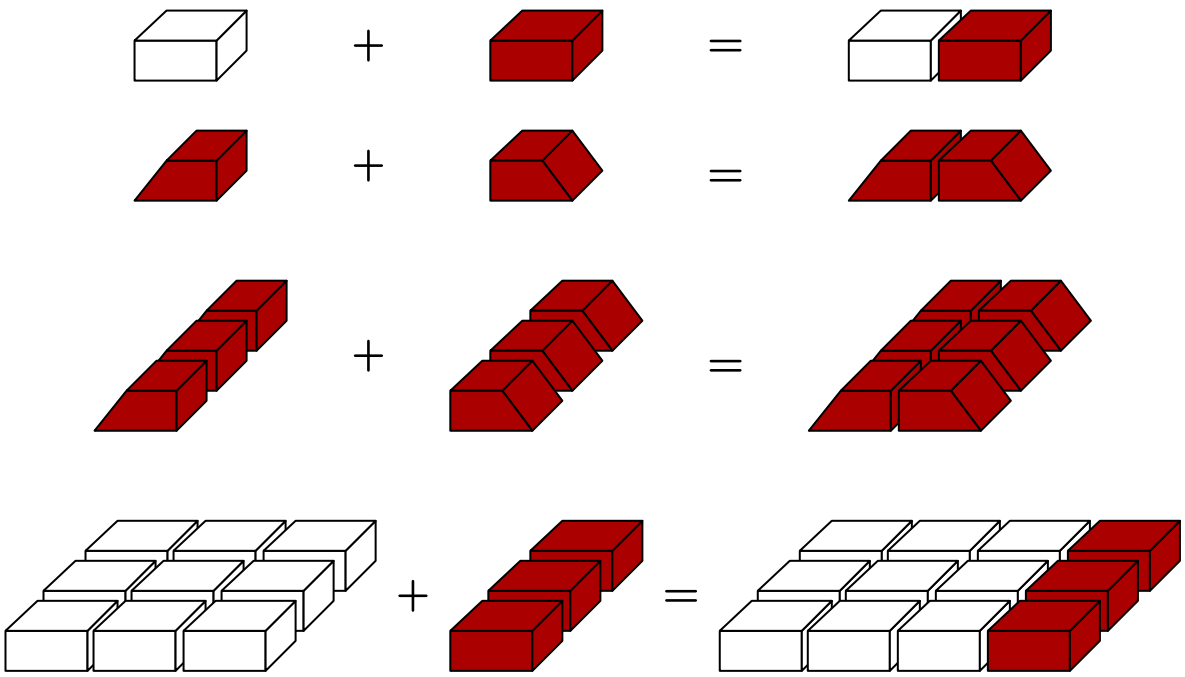


Pour *simplifier* au niveau de cet opérateur (ce qui est décrit ici s'écrit simplement avec 1 seul test) : si la hauteur de `b` (élément ajouté derrière) est plus petite que celle de `a`, alors on ne fait rien (`a` n'est pas modifié) ; si, par contre, elle est plus grande, alors on n'ajoute que la partie de même hauteur que `a`, le reste de `b` étant ignoré.

Voici quelques illustrations de ce principe :



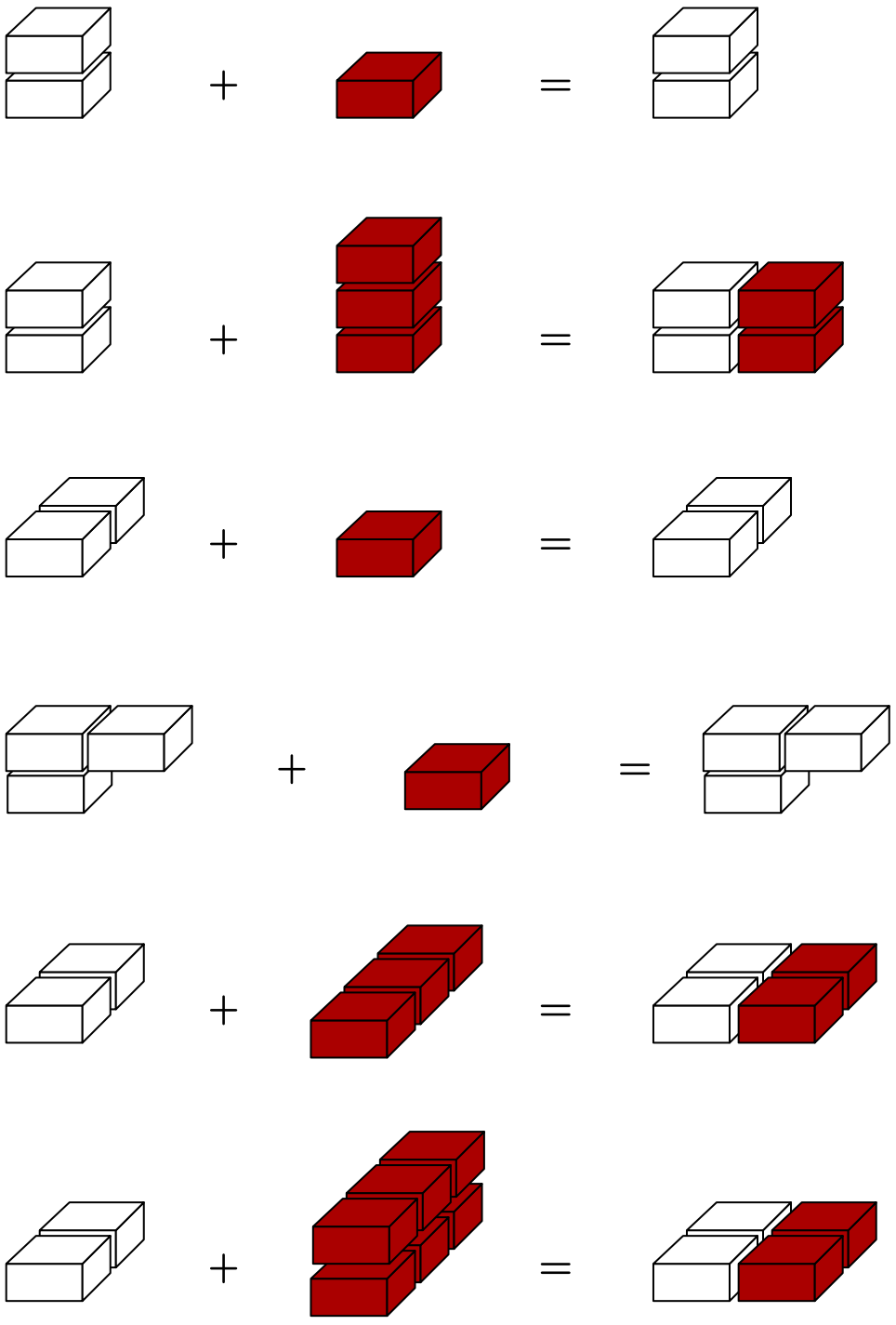
4. l'opérateur interne `operator+=` et l'opérateur externe `operator+` qui ajoutent une `Construction` à droite :



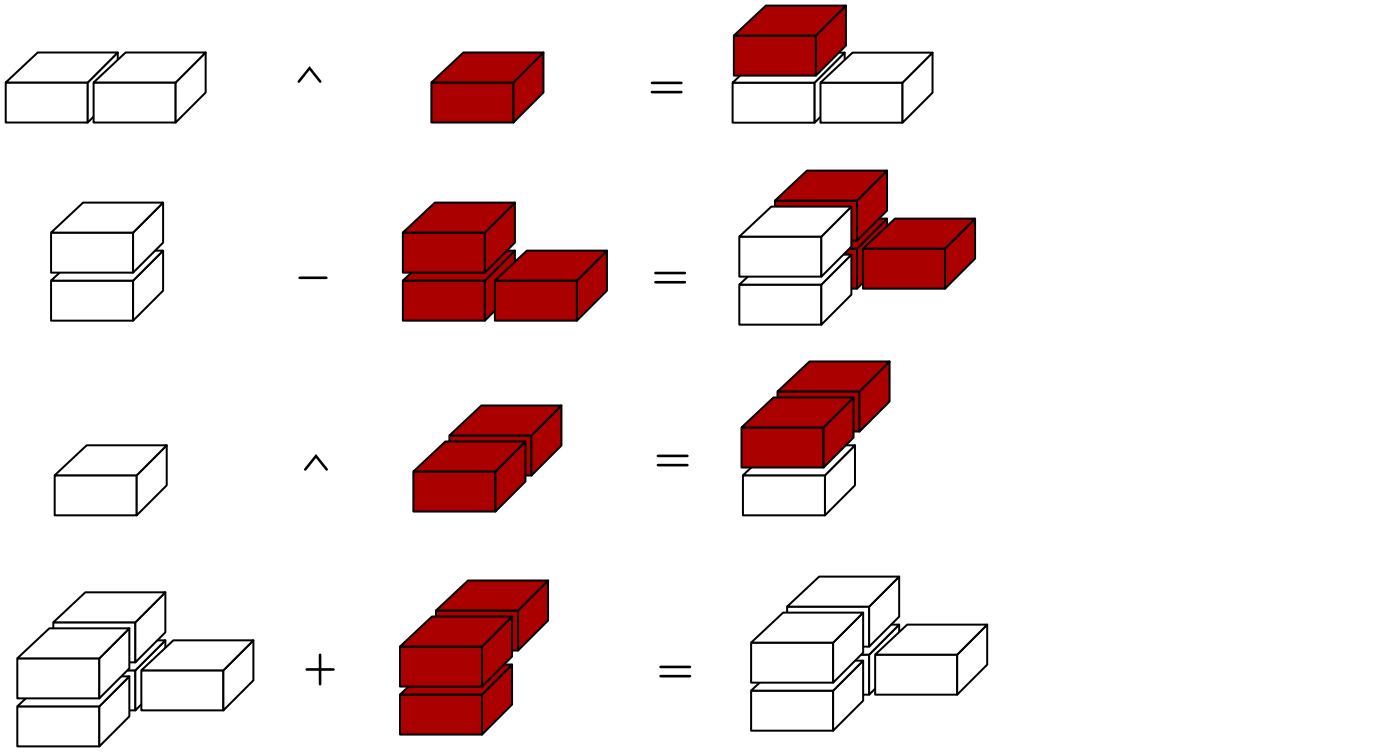
Pour simplifier au niveau de cet opérateur dans le même esprit que l'opérateur `-=` (ce qui est décrit ici s'écrit simplement avec 2 tests séparés) :

— si la hauteur de `b` (élément ajouté à droite) est plus petite que celle de `a`, alors on ne fait rien (`a` n'est pas modifié) ; si, par contre, elle est plus grande, alors on n'ajoute que la partie de même hauteur que `a`, le reste de `b` étant ignoré ;

— si la profondeur de `b` est plus petite que celle de `a`, alors on ne fait rien (`a` n'est pas modifié) ; si, par contre, elle est plus grande, alors on n'ajoute que la partie de même profondeur que `a`, le reste de `b` étant ignoré.



Voici encore quelques exemples généraux :



5. les trois opérateurs suivants :

```
const Construction operator*(unsigned int n, Construction const& a);
const Construction operator/(unsigned int n, Construction const& a);
const Construction operator%(unsigned int n, Construction const& a);
```

permettant de répéter facilement les opérations précédentes :

- `n * a` est la même chose que `a + a + ... + a`, avec `a` répété `n` fois ;
- `n / a` est la même chose que `a ^ a ^ ...^ a`, avec `a` répété `n` fois ;
- `n % a` est la même chose que `a - a - ... - a`, avec `a` répété `n` fois.

Des exemples d'utilisation de ces opérateurs sont donnés dans le `main()` fourni dont le déroulement est explicité plus bas.

2.2 Méthodologie

Nous vous conseillons de procéder progressivement, par ordre, *en testant votre code à chaque étape* :

1. commencer par la classe `Brique` (et la tester) ;
2. faire la base de la classe `Construction`, y compris constructeur et opérateur d'affichage ;
3. commencer simplement par la surcharge interne de l'opérateur `operator^=` qui n'ajoute simplement qu'une dernière couche au dessus ; le tester d'abord avec simplement deux « briques de base » ;
4. procéder ensuite à l'ajout (et au test) de l'opérateur qui ajoute sur la 2e dimension (i.e. `operator-=`) ;
5. puis celui qui ajoute sur la 3e dimension (`operator+=`) ;
6. passer ensuite à leurs surcharges externes correspondantes (opérateurs `operator^`, `operator-` et `operator+`) ;
7. terminer enfin avec les « versions multiples » (opérateurs `operator/`, `operator%` et `operator*`) ;
8. vérifier que le `main()` fourni donne les résultats attendus.

2.3 Exemples de déroulement

L'exemple de déroulement ci-dessous correspond au programme principal fourni.

```
Couche 4 :
                  (obliqueG, rouge) (obliqueD, rouge)
                  (obliqueG, rouge) (obliqueD, rouge)
                  (obliqueG, rouge) (obliqueD, rouge)
Couche 3 :
(obliqueG, rouge) ( pleine , rouge) ( pleine , rouge) (obliqueD, rouge)
(obliqueG, rouge) ( pleine , rouge) ( pleine , rouge) (obliqueD, rouge)
(obliqueG, rouge) ( pleine , rouge) ( pleine , rouge) (obliqueD, rouge)
Couche 2 :
( pleine , blanc) ( pleine , blanc) ( pleine , blanc) ( pleine , blanc)
( pleine , blanc) ( pleine , blanc) ( pleine , blanc) ( pleine , blanc)
( pleine , blanc) ( pleine , blanc) ( pleine , blanc) ( pleine , blanc)
Couche 1 :
( pleine , blanc) ( pleine , blanc) ( pleine , blanc) ( pleine , blanc)
( pleine , blanc) ( pleine , blanc) ( pleine , blanc) ( pleine , blanc)
( pleine , blanc) ( pleine , blanc) ( pleine , blanc) ( pleine , blanc)
Couche 0 :
( pleine , blanc) ( pleine , blanc) ( pleine , blanc) ( pleine , blanc)
( pleine , blanc) ( pleine , blanc) ( pleine , blanc) ( pleine , blanc)
( pleine , blanc) ( pleine , blanc) ( pleine , blanc) ( pleine , blanc)
```