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

Third assignment
Operator overloading

J.-C. Chappelier & J. Sam

1 Exercise 1 — Chemicals

A chemist needs your help to model the *flacons* (which means *vials* in French) of the chemicals that he uses.

1.1 Description

Download the source code available at the course webpage and complete it according to the instructions below.

WARNING: you should modify neither the beginning nor the end of the provided file. It's thus mandatory to proceed as follows:

1. save the downloaded file as `chimie.cc` or `chimie.cpp`;
2. write your code between these two provided comments:

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

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

3. save and test your program to be sure that it works properly; try for instance the values used in the example given below;

4. upload the modified file (still named `chimie.cc` or `chimie.cpp`) in "My submission" then "Create submission".

The supplied code :

- starts with the declaration of the class `Flacon`, described below ;
- then it defines a utility function `afficher_melange` (which means display_mix in french) and finally, in the `main()`, it declares multiple « Flacons » and displays their mixes.

The body of the class `Flacon` is missing and you need to provide it.
A « Flacon » is characterized by :

- the *name* of the product it contains, e.g. « Acide chlroydrique » ;
- its *volume* (in ml), of type `double` ;
- and its *pH* (of type `double`).

The class `Flacon` will also include : :

- a constructor that initializes the attributes using values passed as parameters, in the same order as shown in the `main()` that is given; there should not be any default constructor;
- a method `etiquette`, of prototype
  ```
  ostream& etiquette(ostream& sortie) const ;
  ```
  which displays, in the stream (`ostream`) that is passed as an argument, the « etiquette » (i.e. the label) that should be placed on the « flacon ». This label specifies the name, the volume and the pH of the content of the « flacon » according to the following displaying format (other examples are also presented in the execution example provided further below) :

  ```
  Acide chlroydrique : 250 ml, pH 2
  ```
- an overload of the display terminal << for this class; this overload should use the previous method `etiquette`.

Finally, it should also be possible to mix two « flacons » using the operator + : given two `Flacons` with respective names, volumes and pHs : `name1`, `volume1`, `ph1` and `name2`, `volume2`, `ph2` ; the mix of these two flacons using the operator + gives a new « flacon » for which :

1. the name is « *name1* + *name2* », for example : « Eau + Acide chlorhydrique » ;
2. the volume is the sum of `volume1` and `volume2`;
3. and the pH is (let the chemists forgive us for this gross approximation !) :

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

   The function $\log_{10}$ is written `log10` in C++ and for making the calculation 10^{-x} use the expression `pow(10.0, -x)`.

An example output of the code is provided further below. For this exercise, there will be a 5-point bonus for those who will also provide the class `Flacon` that has an internal operator `operator+=` which allows the mix of the « flacon » itself: even if this is not realistic according to Physics (that is why we use quotation marks), such a « flacon » may see its volume increase as described above ; meaning : after `a += b;` ; the volume `a` is indeed modified (as it will be the case for its name and pH). If you are doing this and would like to ask for the bonus (i.e. be graded out of 35 points), just add the following line in your file :

```
#define BONUS
```

(otherwise, you will be graded out of 30 points).

1.2 Execution examples

```
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 Exercise 2 — Building

The purpose of this exercise is to write a program able to construct "buildings" based on "building blocks" using C++ operators.

This exercise is simpler than it may first seem. Don't go *too* far but follow the proposed (section 2.2).

2.1 Description

Download the source code available at the course webpage and complete it according to the instructions below.

WARNING: you should modify neither the beginning nor the end of the provided file. It's thus mandatory to proceed as follows:

1. save the downloaded file as `construction.cc` or `construction.cpp`;
2. write your code between these two provided comments:

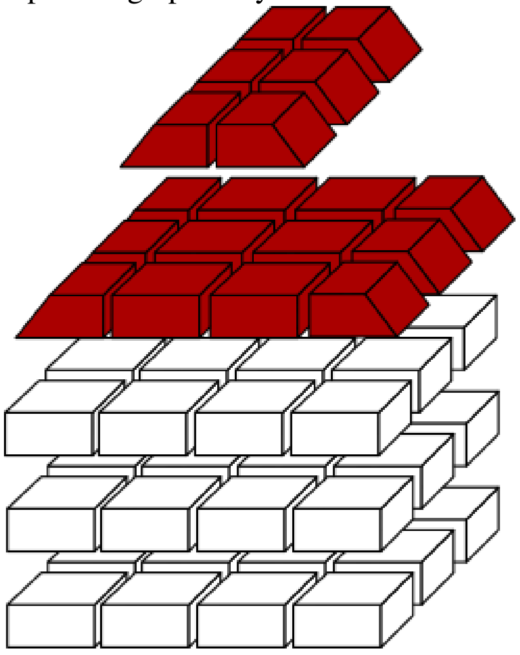
```
/*******************************************
 * Complétez le programme à partir d'ici.
 *******************************************/

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

3. save and test your program to be sure that it works properly; try for instance the values used in the example given below;
4. upload the modified file (still named `construction.cc` or `construction.cpp`) in “My submission” then “Create submission”.

The provided code

- declares the types `Forme` (English: shape) and `Couleur` (English: color) which are useful for the "building blocks"; for simplicity, we use the type `string`;
- starts the declaration of the class `Brique` (English: brick), described further below, which is used for representing the "building blocks";
- inside `main()`, creates multiple "building blocks" and creates a "building" which we can represent graphically like this:

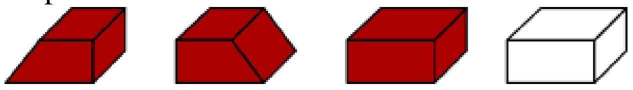


An example of output for the code is provided further below.

The end of the definition of class `Brique` and the definition of class `Construction`, described below, are missing and you are asked to provide them.

2.1.1 The class `Brique`

The class `Brique` represents the "building blocks" of our buildings, as shown in the following example:



Every `Brique` is characterised by a shape and a color, as defined in the provided `construction.cc` file.

You must complete the definition of the class `Brique` by:

- adding a constructor taking two parameters of type `Forme` and `Couleur` (in this order); there is *no* default constructor for this class;
- adding a public method with the protoype

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

  which prints the content of the object using the format:

  - if the color is not an empty string:

    ```
    (shape, color)
    ```

    for example (other examples are provided further below):

    ```
    (obliqueG, rouge)
    ```

  - if the color is an empty string, it displays only the shape (without anything else);
- overloading the printing operator << for this class; the implementation should use the method `afficher`.

2.1.2 The class **`Construction`**

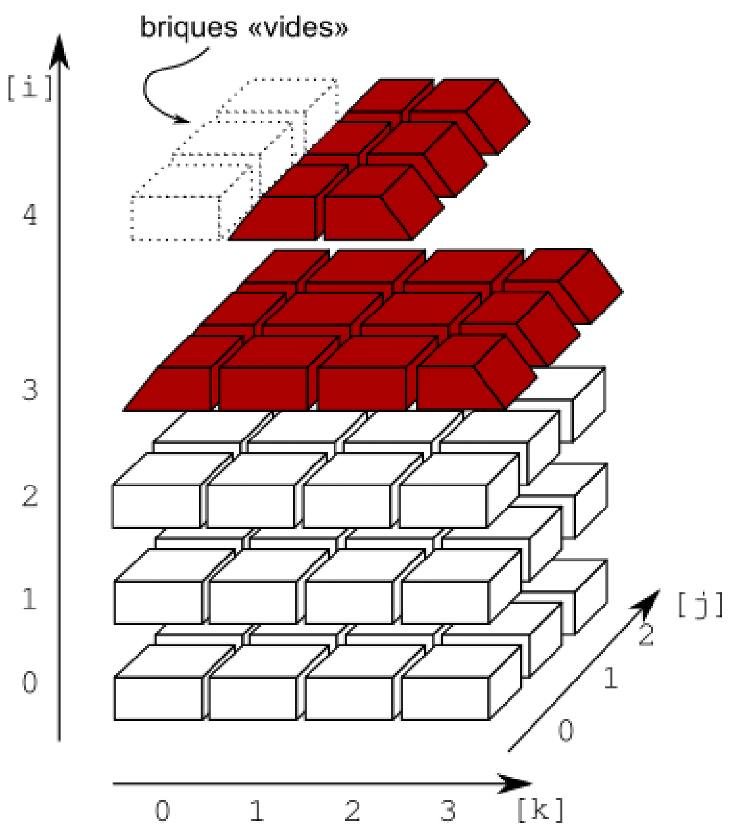
The class `Construction` represents the “buildings” in our program. A “building” is a collection of “building blocks” placed in a 3-dimensional space.

This class must have an attribute `contenu` (respect strictly this name) which is a 3-dimensional dynamic array of “building blocks”, i.e. a `vector` of `vector` of `vector` of `Brique`.

To ease automatic grading, **YOU MUST START THE CLASS `Construction` IN THE FOLLOWING WAY:**

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

The order to internally represent the 3 dimensions (from what direction you look at the building) is shown here:



This means that the first index (i above) represents the heigth, the second (j) the depth and the third (k) the width.

Afterwards, add to the class `Construction`:

- a constructor taking as parameter a `Brique` and creating the `contenu` as an 1x1x1 array containing only this brick;
- a public method with the prototype

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

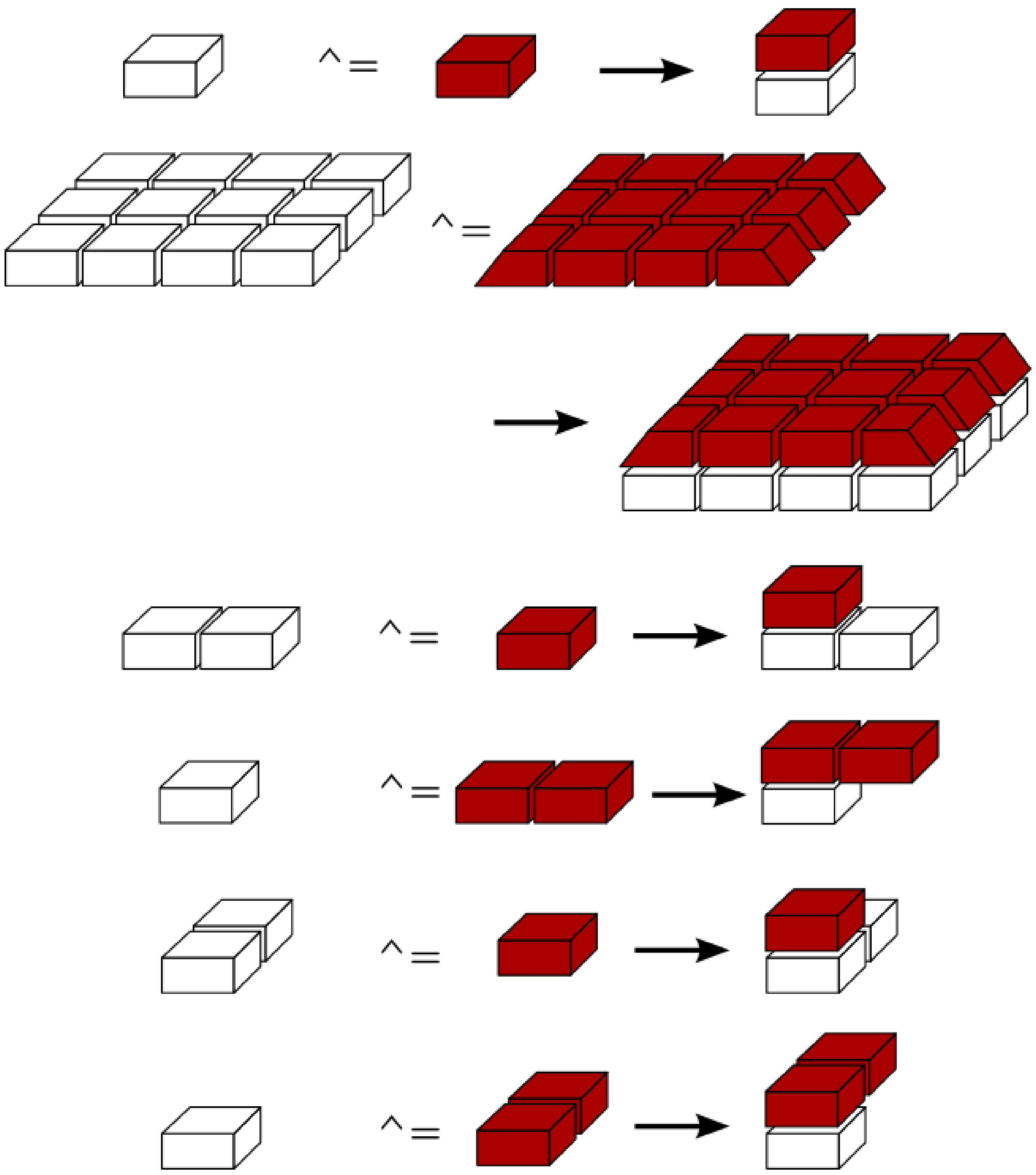
  displaying the building's content layer by layer as shown further below in the output example (a message "`Couche` *`numéro`* `:`" (English: Layer number) must be displayed at the beginning of each layer of the construction). If the construction is empty, nothing is displayed.

2.1.3 Operators for the class `Construction`

The following ten operators must be added to the class `Construction`[1]:

[1]Those operators are only used to describe the layers of the construction. It is not their purpose to check whether the corresponding result is physically possible or not.

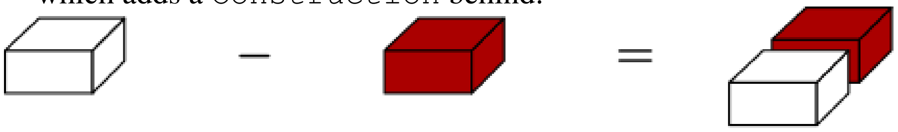
1. the (external) operator for printing << which should use the method `afficher`;
2. the internal operator `operator^=` and the external operator `operator^` which adds a `Construction` on top:
   - `a ^= b;` adds the `Construction` `b` on top of `Construction` `a`, as shown here:

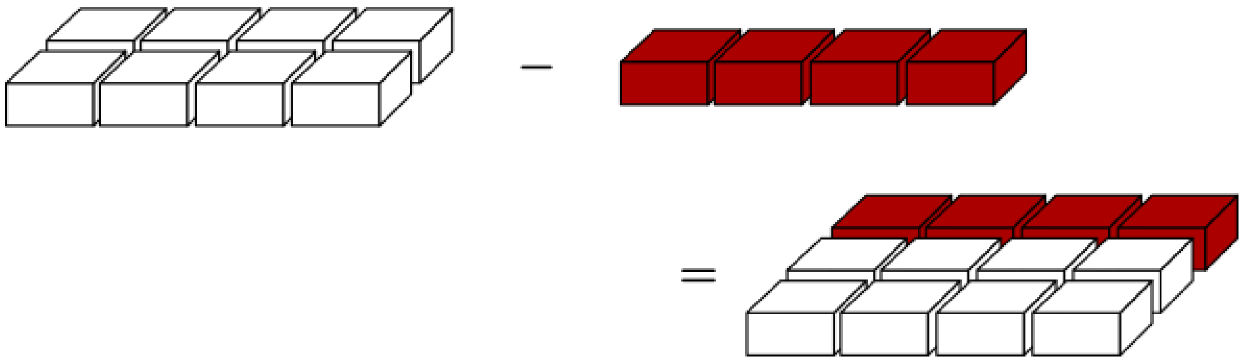


- `a ^ b;` creates a new `Construction` which is the result of `Construction b` placed on top of `Construction a`;

3. the internal operator `operator-=` and the external operator `operator-`

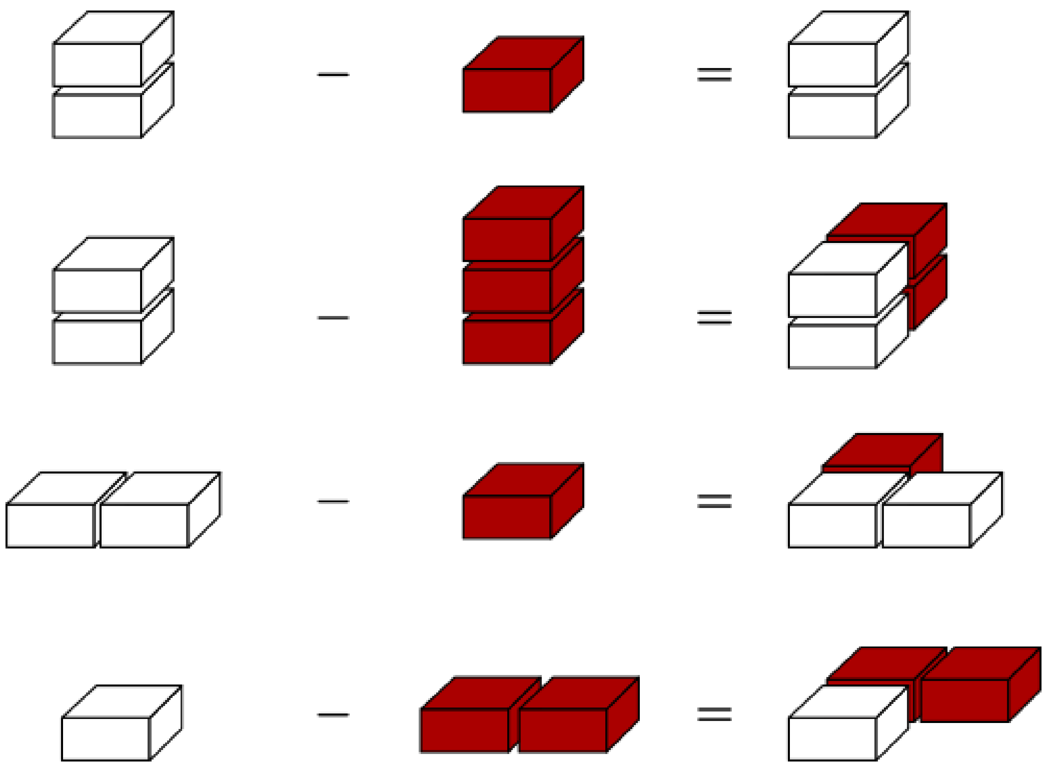
which adds a `Construction` behind:



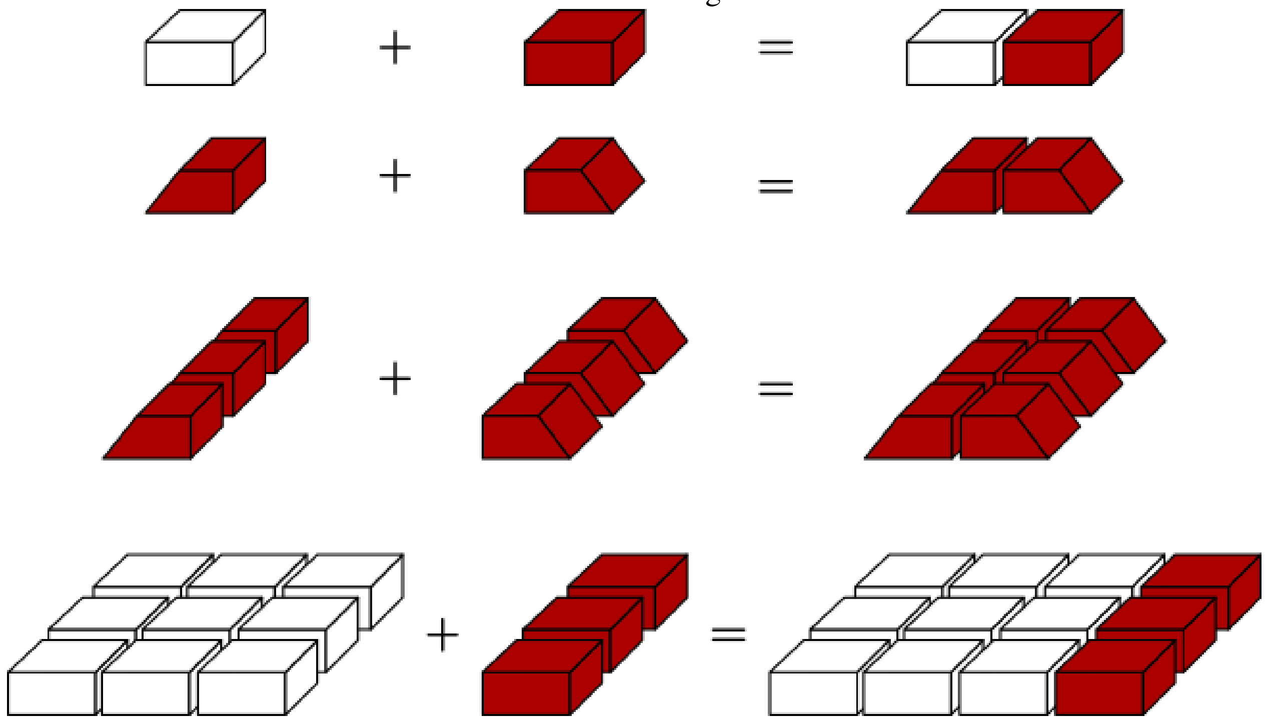


To make it *simple*, this operator does the following tests (this can be implemented with only 1 test): if the height of `b` (element added behind) is smaller than the height of `a`, no action is taken (`a` is not modified); if, by contrast, it is larger, then we only add the part of the same height as `a` and the rest of `b` is ignored.

Here are a few examples:

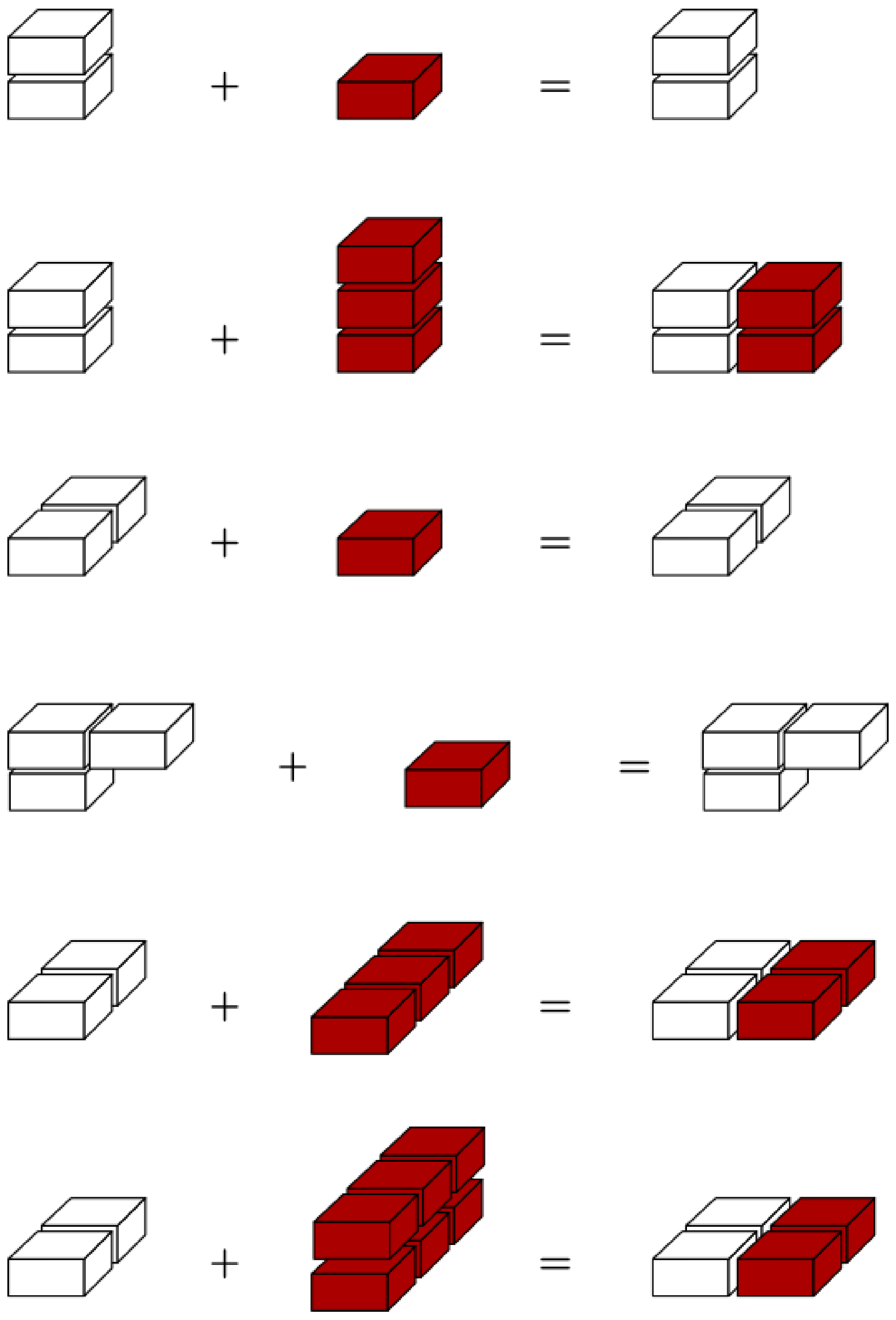


4. the internal operator `operator+=` and the external operator `operator+` which adds a `Construction` to the right:

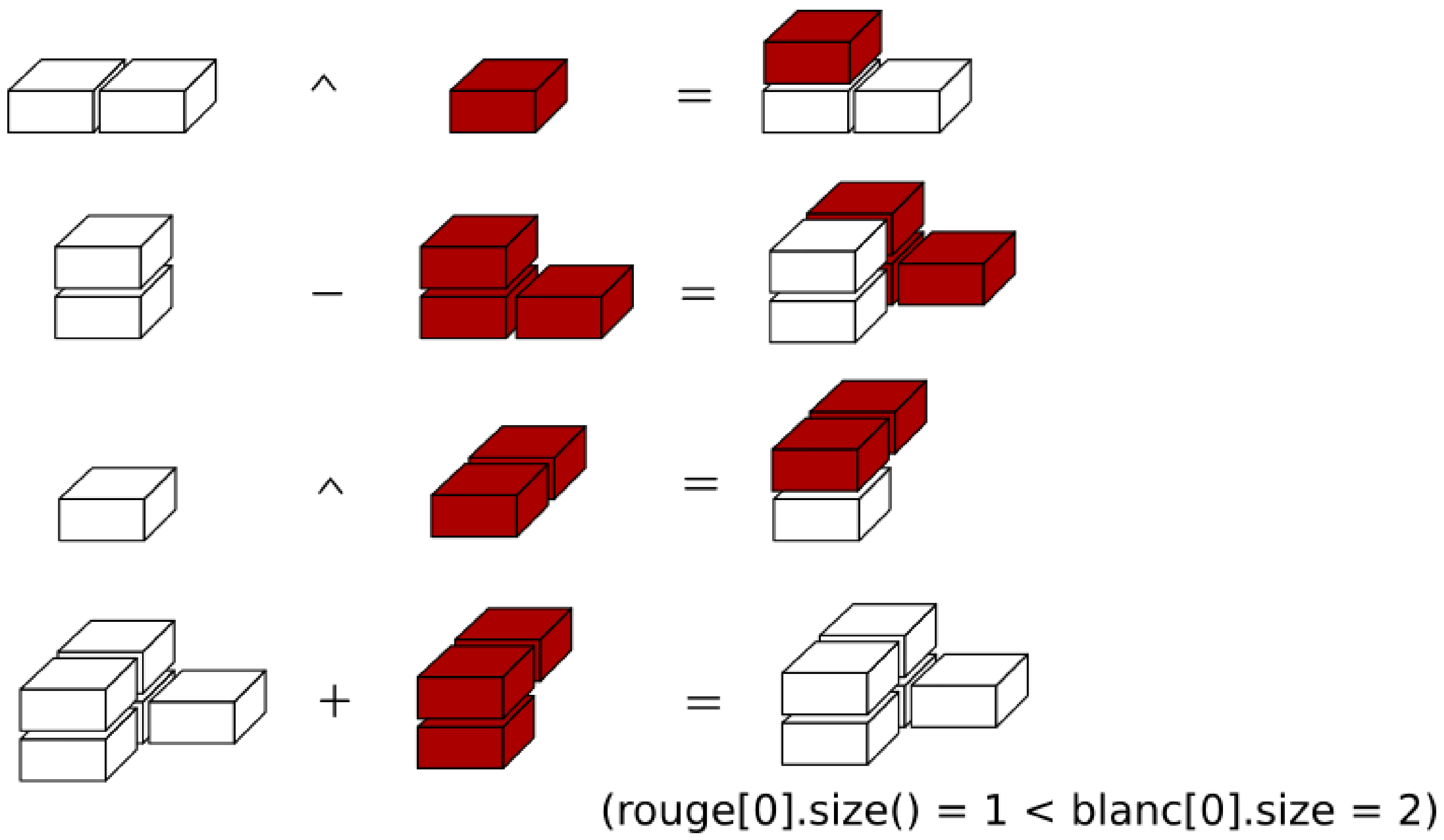


This operator does the following tests (this can be implemented with 2 tests only):

- if the height of `b` (the element added to the right) is smaller than the height of `a`, no action is taken (`a` is not modified); if, by contrast, it is higher, then we only add the part of the same height as `a`, the rest of `b` is ignored;
- if the depth of `b` is smaller than the depth of `a`, no action is taken (`a` is not modified); if, by contrast, it is larger, we only add the part of the same depth as `a`, the rest of `b` is ignored.



Here are a few general examples more:



5. the following operators

   ```
   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);
   ```

   allow us to repeat easily some operations:

   - `n * a` is the same as `a + a + a`, with + repeated `n` times;
   - `n / a` is the same as `a ^ a ^ a`, with ^ repeated `n` times;
   - `n % a` is the same as `a - a - a`, with − repeated `n` times .

Examples of using these operators are given in the provided `main()` and the output is also shown further below.

2.2 Methodology

We suggest that you work carefully, step by step, *testing your code after each step*:

1. start with the class `Brique` (and test your implementation);
2. write the basics of the class `Construction`, including the constructor and the printing operator;
3. start by overloading the internal operator `operator^=` that simply adds a layer of blocks above the building; test it right away with two simple "basic blocks";
4. next, add and test the operator `operator-=` that adds building blocks in the 2nd dimension;
5. then add the one that works in the 3rd dimension (that is `operator+=`);
6. continue with overloading their external counterparts (operators `operator^`, `operator-` and `operator+`);
7. finally, add the "repeated operators" (operators `operator/`, `operator%` and `operator*`);
8. verify that the `main()` gives the expected results.

2.3 Execution examples

The example of output below corresponds to the provided program.

```
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)
```