Vous avez 666 allumettes que vous voulez répartir par boites de 13, combien de boites pleines y aura-t-il ? S'il y a une boite non-pleine combien d'allumettes contiendra-t-telle ?

Ces questions sont liées à ce qu'on appelle la division euclidienne (ou division entière), c'est-à-dire qu'on souhaite trouver *nbBoites* et *nbReste* tels que :

- 666 = 13 * *nbBoites* + *nbReste*
- 0 <= *nbReste* < 13

La première condition signifie qu'on ne perd aucune allumette et la seconde que la boite non-pleine (si elle existe, c'est à dire si *nbReste* est différent de 0) ne peut pas contenir 13 allumettes (elle serait pleine sinon).

En C++ il est possible de calculer *nbBoites* et *nbReste* très facilement, à l'aide de deux nouveaux opérateurs :

```
int nbBoites = 666 / 13;
int nbReste = 666 % 13;
cout << nbBoites << endl;
cout << nbReste << endl;
```

↳
```
51
3
```

En terme de division euclidienne on a donc que, pour *a* et *b* deux entiers :

- `a / b` donne le *quotient* de la division euclidienne de *a* par *b*
- `a % b` donne le *reste* de la division euclidienne de *a* par *b*

Que se passe-t-il si au lieu de 666 on avait choisi un dividende négatif (on supposera que le diviseur est positif) ?

On ne peut plus parler d'allumettes mais imaginons qu'on doive payer 666 euros pour une anthologie de Heavy Métal (42 DVD !) et qu'on n'ait que des billets de 50 euros. On va donc donner 700 euros (soit 14 billets) et on va nous rendre 34 euros . Comme on paye quelque chose on a moins d'argent dans notre portefeuille, donc on perd de l'argent :

- -666 = 50 * (-14) + 34
- 0 <= 34 < 50

On a donc perdu 14 billets de 50 mais on a récupéré 34 euros.

Si on essaie d'écrire cela en C++, on a :

```
cout << -666 / 50 << endl;
cout << -666 % 50 << endl;
```

↳
```
-13
-16
```

Ce n'est pas ce qu'on voulait ! La division entière de C++ ne correspond à la division euclidienne que pour des dividendes positifs. Pour garantir :

- *dividende* = *diviseur* * *quotient* + *reste*
- 0 ≤ *reste* < *diviseur*

il faudra écrire, en supposant que *diviseur* > 0 :

```
int quotient = dividende / diviseur;
```

```
int reste = dividende % diviseur;
if (reste < 0)
{
   quotient = quotient - 1;
   reste = reste + diviseur;
}
```

Un exemple fréquent d'utilisation de ces opérateurs :

```
if ((nombre % 2) == 0)
{
   cout << "Le nombre est pair";
}
```

En effet, si le reste vaut zéro c'est que *nombre* = *2 x quotient* donc *nombre* est divisible par 2.