Zip et unzip

Intentionnalités inspirées de Python (et autres langages, probablement. Cf https://docs.python.org/3.3/library/functions.html#zip). Permet de manipuler des listes.

Zip : prend plusieurs listes et en crée une seule.

```
const auto x = { 1, 2, 3 };
const auto y = { 'a', 'b', 'c' };
const auto z = zip(x, y);
std::cout << z << std::endl; // affiche ((1,a), (2,b), (3,c))
```

Unzip : prend une liste et la sépare en plusieurs listes.

```
const Zip z = { { 1, 'a' }, { 2, 'b' }, { 3, 'c' } };
styd::tie(x, y) = unzip(z);
std::cout << x << std::endl; // affiche (1, 2, 3)
std::cout << y << std::endl; // affiche (a, b, c)
```

Vérifier les données

Écrire une fonction qui vérifie que deux vectors ont le même nombre d'éléments, en utilisant un algorithme de la bibliothèque standard.

```
bool check_zip(vector<int> const& x, vector<char> const& y);
```

Avec un algorithme standard

Écrire une fonction qui zip deux vectors, en utilisant un algorithme de la bibliothèque standard.

```
vector<pair<int, char>> zip1(vector<int> const& x, vector<
```

```
char> const& y);
```

Avec des boucles

Idem, en utilisant une boucle for sur itérateur (zip2a), une boucle for avec indices (zip2b). Idem avec while (zip2c et zip2d).

Quelle est la syntaxe que vous préférez et pourquoi ?

Est-il possible d'utiliser un “range-based for” ?

Générique sur les types

Écrire une fonction qui est générique pour les types.

```
vector<pair<T, U>> zip3(vector<T> const& x, vector<U> const&
y);
```

Générique sur les collections

Écrire une fonction qui est générique pour les collections (en entrée uniquement ?). S'inspirer des algorithmes de la bibliothèque standard.

```
void zip3(InputIterA first_x, InputIterA last_x,
          InputIterB first_y, InputIterB last_y,
          OutputIter first_z);
```

Adaptateur

La fonction de la question précédente n'a pas la même syntaxe que les fonctions des questions avant (vector en paramètre mutable plutôt qu'en retour de fonction), par homologie avec les algorithmes standards (et pour éviter la copie).

Écrire un **adaptateur** permettant de convertir le zip de la question

précédente (donc sans réécrire le zip).

```
vector<pair<T, U>> z = (...) zip3(x1, x2, y1, y2, z1) (...);
```

Paire d'itérateurs (Range)

Utiliser des paires d'itérateurs sur chaque collection en paramètre, plutôt que des itérateurs (sauf pour la sortie).

```
void zip5(std::pair<InputIterA, InputIterA> x,
          std::pair<InputIterB, InputIterB> y,
          OutputIter first_z);
```

Paire d'itérateurs

Idem, mais chaque paire représente le début ou la fin dans les deux collections.

```
void zip5b(std::pair<InputIterA, InputIterB> first,
           std::pair<InputIterA, InputIterB> last,
           OutputIter first_z);
```

Quelle syntaxe vous semble la plus pratique ?

Est-il possible d'écrire zip5 et zip5b en utilisant zip3 et des adaptateurs ?

Meta-programmation

Prendre un nombre indéfini de collections en entrée. Plusieurs approches sont possibles :

- réécrire zip5 avec std::tuple ;
- réécrire zip5b avec std::tuple ;
- prendre un seul std::tuple en paramètre, contenant successivement le début et la fin de chaque collection ;
- en utilisant un variadic template.

Écrire chaque approche. Laquelle préférez-vous ?

Note : cette question sort des limites de ce cours.

Chapitre précédent	Sommaire principal	Chapitre suivant

Cours, C++