Projet n°2 : Tuple

Un `tuple` est un objet classique dans les langages de programmation. Il s'agit d'une classe utilitaire qui contient un nombre quelconque de valeurs dont les types peuvent être différents.

Le but de ce projet sera d'implémenter une classe `Tuple`, librement inspirée de la classe `std::tuple` que propose la librairie standard.

Afin de pouvoir fonctionner correctement, les types stockés dans un `tuple` doivent être au moins déplaçables et avoir un constructeur par défaut. Si ce n'est pas le cas, cela entraînera une erreur de compilation. *Ce n'est pas à vous de gérer ce cas de figure.* Dans ce projet, beaucoup de cas d'erreur seront gérés par le compilateur lui-même et ne relève pas de l'implémentation de la classe.

Si les messages d'erreur de `gcc` vous paraisse peu explicite, vous pouvez tester votre projet avec `clang`. Voici la commande pour configurer `cmake` avec `clang`.

```
CXX=clang++ C=clang cmake -B build-clang
```

Étape 1 : Création et récupération

La première étape est donc de pouvoir instancier un `Tuple` et d'accéder à ces éléments. La classe `Tuple` dispose donc d'un constructeur par défaut et d'un second constructeur qui initialise tous les éléments du `tuple`.

Une autre façon de créer un `tuple` est de faire appel à la fonction `t::makeTuple()`. La fonction doit déduire automatiquement le type du `tuple` en fonction des paramètres qu'on lui passe.

Pour récupérer les valeurs d'un `tuple`, on utilise la méthode `Tuple::get()`. Cette méthode est templatée, et elle s'attend à avoir au minimum le numéro de l'élément à récupérer. La méthode existe en deux versions. La première qui est `const` et qui doit renvoyer une référence constante sur l'objet stocké. La seconde qui n'est pas `const` et qui doit renvoyer une référence sur l'objet stocké.

Étape 2 : Opérateurs de comparaison

La classe `Tuple` dispose également de tous les opérateurs de comparaison : `==`, `!=`, `<=`, `<`, `>=`, `>`. Pour comparer deux `tuples`, on compare les éléments deux à deux en partant du premier jusqu'au dernier (ordre lexicographique).

Étape 3 : Opérateurs arithmétiques

Notre classe `Tuple` dispose des opérateurs arithmétiques habituels (`+`, `-`, `*` et `/`) ainsi que leur équivalant d'affectation (`+=`, `-=`, `*=` et `/=`). L'utilisation de ces opérateurs signifie qu'on applique élément par élément l'opérateur correspondant. Cela implique que les deux `tuples` doivent avoir le même nombre d'éléments et que ceux-ci doivent être compatibles entre eux. Il est possible, par exemple, d'additionner un `int` et `double` mais pas un `float` et un `std::string`.

Si les types ne sont pas compatibles ou qu'il n'y a pas le bon nombre d'éléments entre les `tuples`, c'est le compilateur qui retourne une erreur et non l'implémentation de la classe (exception). Ce n'est

donc pas à vous de faire cette vérification.

Étape 4 : Concaténation

Il est possible de concaténer deux `tuples`. Le résultat est un troisième `tuple` qui contient tous les éléments des deux `tuples` d'origine. La concaténation s'appelle via l'opérateur `Tuple::operator|()`.

Exemple d'utilisation

```cpp
#include <iostream>

#include "Tuple.h"

int main() {
  tpl::Tuple<int, double, std::string> t(42, 3.14, "The cake is ");

  std::cout
    << t.get<0>() << ", "
    << t.get<1>() << ", "
    << t.get<2>() << std::endl;
  // 42, 3.14, The cake is

  tpl::Tuple<int, double, std::string> t2(-42, -3.14, "a lie");
  t += t2;

  std::cout
    << t.get<0>() << ", "
    << t.get<1>() << ", "
    << t.get<2>() << std::endl;
  // 0, 0, The cake is a lie!

  auto t3 = tpl::makeTuple(10, 10.0f);
  tpl::Tuple<std::size_t, double> t4(2lu, 2.0);
  auto t5 = t3 * t4;
  std::cout << t5.get<0>() << ", " << t5.get<1>() << std::endl;
  // 20, 20

  auto t6 = std::move(t) | std::move(t5);
  std::cout
    << t6.get<0>() << ", "
    << t6.get<1>() << ", "
    << t6.get<2>() << ", "
    << t6.get<3>() << ", "
    << t6.get<4>() << std::endl;
  // 0, 0, The cake is a lie!, 20, 20

  return 0;
}
```