C++ (M2)

TD #4

Rédacteur: Stéphane Vialette

Dans ce TD, on écrit une classe pour stocker des n-uplets de valeurs, le type des n valeurs étant donné par une liste de n types.

Exercice 1

- Créer un type `TypeList<T,U>` dans lequel figurent deux définitions de types : `head` égal à `T` et `tail` égal à `U`. On utilisera un type `NullType` pour marquer la fin de la liste. Ainsi `TypeList<int, TypeList<double, NullType> >` représente la liste `[int; double]`.
- Créer une classe `template GetType` paramétrée par une liste de types `L` et un entier positif `N` de sorte que `GetType<L,N>::type` est le type d'indice `N` de la liste `L`.

Exercice 2

Écrire une classe template `Nuple` paramétrée par une liste de types `L` et qui permet de stocker autant de valeurs qu'il y a de types dans la liste `L`, le type de la `K`-ième valeur étant `GetType<L,K>::type`. Proposer une implémentation pour la classe `Nuple<L>` (on écrira les méthodes plus tard), de sorte que `Nuple<L>::val` est la première valeur du `N`-uplet.

Exercice 3

Écrire une classe template `HelpGet` paramétrée par une liste `L` et un entier positif `K` et qui contient une méthode statique `get` dont l'argument est un `N`-uplet et qui retourne une référence sur la `K`-ième valeur du `N`-uplet.

Exercice 4

- a) Écrire une méthode template `at` dans la classe template `Nuple`, paramétrée par un entier positif `K`, sans argument et qui permet d'accéder à la `K`-ième valeur de la liste. On pourra écrire :

```
typedef TypeList<int, TypeList<double, NullType> > liste_t;
Nuple<liste_t> u;
u.at<0>() = 3.4; u.at<1>() = 4.5;
std::cout << u.at<0>()
          << ' '
          << u.at<1>()
          << std::endl; // affiche "3 4.5"
```

– Faire en sorte que l'on puisse spécifier les valeurs d'un N-uplet avec la syntaxe suivante :

```
u << 3.4 << 4.5;
```