TP1: Template

Fonction template

1 Ecrire une fonction distance qui calcule la distance entre 2 valeurs réelles. On utilisera la formule: distance (p,q) = abs(q-p).

2 Transformer la en version générique paramétrée par le type de p et q

Attention:
abs(x) pour les entiers
fabs(x) pour les réels
=> utiliser std::abs(x) qui est générique

Tester avec des int
Tester avec des unsigned int

Remarques:
- déduction des paramètres templates par inférence de type à partir des paramètres lors de l'appel.
- Postfix des constantes numérique entières: l : long / u: unsigned / h short
  - Ex: 453ul (64bit non signé)

Que se passe-t-il (dans 50% des cas) avec notre formule pour 2 variables de type *unsigned*

Pour plus de clarté définir à l'aide de using les types Int32 Uint32 Int64 Uint64

3 Faire une version spécifique *Uint64* (qui sera appelée automatiquement quand il faut)

4 Faire une version pour les chiffres sous la forme de chaine (std::string) qui renvoit un *Uint64*. Utiliser le *for range* sur les string et assert pour vérifier que les chaines contiennent uniquement de chiffres

Traits

5 Faire une classe paramétrée *TInfo* qui fournit les services suivants:
- *usable*: type utilisable pour la fonction distance ? (booléen)
- *name*: nom du type sous la forme d'une std::string

La version générique sera implémentée avec:

- *usable*-> faux
- *name* : utiliser typeid::name()

Spécialiser la classe pour int32 uint32 float double etc...

Utiliser TInfo::usable avec static_assert dans la fonction générique distance
=> Tester !!!

Classe template

On veut écrire une version de distance qui fonctionne sur des vecteurs de dimension et de type quelconque (encapsulation de T[N])

6 Ecrire une classe Vec paramétrée par un entier N et un type T

- constructeur par défaut qui initialise à (O,O,..,O)
- constructeur qui initialise à (x,x,..,x)
- operateur [ ] (const et non const)
- operateur -
- la méthode length
- ...

On utilisera const et & autant que possible

Pb: je veux initialiser mon Vec<3,float> à (0.1, 0.2, 0.3) directement dans le constructeur
Bien-sur je veux pouvoir le faire qqsoit N !

Il faut une fonction à nombre de param variable. => utilisation des template variadic

Template Variadic
7 Ecrire une méthode init à nb de paramètres templates variable.
Rappel : il faut utiliser la récursivité: on initialise la bonne case avec le paramètre v1 et appel la méthode avec le reste des paramètres.

```
template <typename... XX>
void init(const T& v1, XX... vars)
```

Comment choisir la case du tableau à initialiser: indice sizeof...( )
Arrêt de la récursivité : faire une version spécifique de init
Ecrire le constructeur qui utilise init

8 Ajouter un static_assert dans le constructeur pour vérifier le nombre de paramètres

9 Comment faire pour vérifier que tous les paramètres sont du même type (T)
Solution 1: une classe de Trait à nb de param variable dont le *::value* est vrai si tous les types sont identiques.

Solution 2: une fonction constexpr template à nb de param variable qui renvoie vrai si tous les types sont identiques.

-> utiliser avec static assert dans le constructeur

Conversion implicite

ecrire une fonction void affiche(const Vec<3,int>& v)
Que se passe-t-il quand on appel affiche(3);
Utiliser le qualifier explicit pour l'éviter

Tester l'utilisation de decltype