Base du Développement Logiciel

22 janvier 2019

1 Création et affichage de points

Créez une structure de point 4D stockant ces coordonées sous forme de réel simple précision.

Définissez une fonction pour créer un point avec des coordonnées aléatoires comprises entre 0 et 1. Elle aura la signature suivante :

```
point4D randomPoint();
```

Pour ce faire, utiliser les générateur aléatoire fourni par C++11 (cf cppreference).

Définissez une fonction pour afficher les coordonnées d'un point sur le *stream* passé en argument. Elle aura la signature suivante :

```
std::ostream &operator<<(std::ostream &out, point4D const &p);
```

Combinez l'algorithme `std::generate_n`, l'itérateur renvoyé par `std::back_inserter` et la fonction `randomPoint` pour remplir un vecteur de type `points4D` avec 5 points aléatoires. (Note : le code obtenu ne doit pas dépasser une instruction.)

Combinez l'algorithme `std::for_each` avec une fonction anonyme affichant un point sur la sortie standard pour afficher le contenu du vecteur précédent. (Note : le code obtenu ne doit pas dépasser une instruction.)

2 Longueur de chemin

Définissez une fonction calculant la distance euclidienne entre deux points 4D :

```
double dist(point4D const &p1, point4D const &p2);
```

Définissez une fonction `pathLength` prenant un vecteur de type `std::vector<point4D>` en argument et renvoyant la longueur du chemin fermé passant par tous les points de ce vecteur (dans l'ordre du vecteur). Utilisez l'algorithme `std::for_each` pour cela.

Proposez une variante de `pathLength` qui utilise l'algorithme `std::accumulate`.

3 Échantillonage

Définissez une fonction `quantize` qui prend un point en argument et renvoie un point dont les coordonnées ont été arrondies par défaut à 0.1 près. Par exemple, pour le point $(0.234, 0.456, 0.678, 0.254)$, la fonction doit renvoyer $(0.2, 0.4, 0.6, 0.2)$.

Définissez un opérateur de comparaison stricte `lt` qui prend deux points 4D en argument et effectue une comparaison lexicographique stricte sur leurs versions arrondies par `quantize`.

Essayez de définir un type d'ensemble de points ordonnés par `lt` de la façon suivante :

```
using point_set = std::set<point4D, lt>;
```

Malheureusement, le langage C++ n'autorise pas d'utiliser directement une fonction comme deuxième paramètre template de `std::set`. Définissez donc une classe ayant une unique fonction membre template `operator()` qui appelle `lt` pour comparer deux points et utilisez cette classe pour définir `point_set`.

Créez un vecteur de 10,000 points aléatoires. Combinez l'algorithme `std::copy` et la fonction `std::inserter` pour transformer le vecteur en un ensemble ordonné de type `point_set`. Affichez la taille de l'ensemble résultant.