ensIIE
Ecole Nationale Supérieure d'Informatique pour l'Industrie et l'Entreprise

TP STL
(noté)

Prérequis

- Chaque exercice se présente sous la forme de classes C++ à compléter dont vous pourrez trouver un squelette d'implémentation dans /pub/FISE_LAOA24/TPs/TP1/TP_STL-<date>.zip
- Référence C++ (& STL) :
  - http://www.cplusplus.com/reference/
  - https://en.cppreference.com/

1. Étude de code

Notez les `using` au début du main qui permettent de définir la nature des éléments des conteneurs (`NumType = double`), le conteneur utilisé (`Container = vector<NumType>`), ainsi que la nature de l'output iterator défini au début du `main` pour afficher le contenu d'une étendue d'itérateurs sur la console (`Printer = ostream_iterator<NumType>`).

Etudiez le patron de fonction

```
template <typename Iterator>
    requires input_iterator<Iterator>
bool mysteryCheck(const Iterator & first, const Iterator & last)
{
  …
}
```

défini au début de main.c.

Que fait ce patron de fonction ?

2. Génération de valeurs aléatoires bornées [min … max]

Créez ou complétez la classe foncteur `BoundedRandomGenerator<Arithmetic>`, dont l'opérateur d'appel de fonction (`Arithmetic operator()(void)`) permet de générer des nombres aléatoires de type `T` compris entre deux bornes (min et max). On pourra utiliser : (voir man 3 random)

- `void srandom(unsigned seed)` pour initialiser le générateur de nombres aléatoires lors de la construction de l'objet. On pourra avantageusement utiliser `time(nullptr)` comme `seed`.
- `long random()` pour générer un nombre aléatoire (plus aléatoire que la fonction `rand()`[1]) compris entre 0 et RAND_MAX. Puis il faudra renormaliser ce nombre entre min et max et le caster en `T`.

2.1. Dans le programme principal (main.cpp), utilisez un algorithme standard avec ce foncteur pour remplir le `Container v`[2] avec des valeurs aléatoires comprises entre -12.3 et +25.7.

2.2. Affichez le contenu de `v` en utilisant un algorithme standard avec l'`ostream_iterator<NumType> printer` défini au début du main.

[1] mais non présent sur les OS Windows sur lesquels il faudra donc utiliser `srand` & `rand`.

[2] Où `Container v` est un `vector<double>` de `nbElts` initialisé avec la valeur `defaultVal`.

3. Calcul de la valeur moyenne et de l'écart type

Créez ou complétez la classe foncteur `class StatFunctor<Arithmetic>` dont l'opérateur d'appel de fonction (`void operator()(const Arithmetic & value)`) permet d'additionner des valeurs et des valeurs au carré de type `T` en vue de calculer sur une série de n valeurs $X = \{x_1, x_2, \cdots, x_n\}$:

- la moyenne $\bar{x} = E(X) = \frac{1}{n}\sum_{i=1}^{n} x_i$: avec une méthode `double mean() const`
-
- et l'écart type $\sigma = \sqrt{E(X^2) - E(X)^2}$: avec une méthode `double std() const`

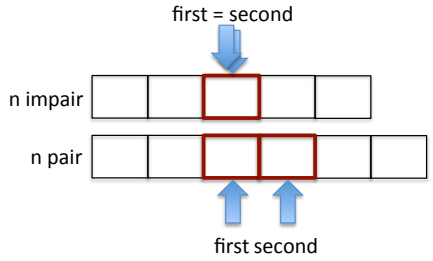
3.1. Appliquez ce foncteur sur l'ensemble des éléments de `v` avec un algorithme standard, puis affichez la moyenne et l'écart type sur la console.

4. Recherche de la médiane

Soit E un ensemble de n nombres triés par ordre croissant, la médiane de E est son élément central, ou plus exactement :

$$\begin{cases} \text{si } n \text{ est impair } médiane = \left(\frac{n+1}{2}\right)^{ème} élément \\ \text{si } n \text{ est pair } médiane = \left(\left(\frac{n}{2}\right)^{ème} élément + \left(\frac{n}{2}+1\right)^{ème} élément\right)/2 \end{cases}$$

Nous souhaitons écrire deux algorithmes de recherche de la (ou les) position(s) des éléments constituant la médiane (`median` et `median2`). Pour ce faire, ces algorithmes renverront une paire d'itérateurs indiquant la (ou les) positions où trouver le (ou les) élément(s) constituant la médiane. Dans une étendue d'itérateurs sur des éléments triés on aura alors pour la paire[3] :



Alors que pour une étendue d'itérateurs sur des éléments non triés, first & second peuvent se trouver n'importe où dans l'étendue d'itérateurs.

4.1. Créez une copie triée de `v` dans `vt` en utilisant un algorithme standard. Puis affichez ses éléments.

4.2. Complétez l'algorithme
```
pair<Iterator, Iterator>
median(const Iterator & first,
       const Iterator & last)
```
pour trouver une paire d'itérateurs sur les éléments constituants la médiane d'une étendue d'itérateurs [first, last) dont les éléments sont censés être triés (il faudra le vérifier).
Si l'étendue [first, last) n'est pas triée ou si elle est vide on renverra la paire <last, last>.

4.3. Utilisez cet algorithme pour trouver la paire d'itérateurs de la médiane de `vt`, puis affichez le ou les éléments constituant la médiane, et sa valeur. Dans la mesure où l'on n'utilisera pas la paire

[3] Attention first & second sont ici les membres de la paire. A ne pas confondre avec [first, last) d'une étendue d'itérateurs.

d'itérateurs résultante pour modifier `vt`, on se contentera de **`const_iterator`**(s) lors de l'appel de `median`.

4.4. Complétez l'algorithme
```
pair<Iterator, Iterator>
median2(const Iterator & first,
        const Iterator & last)
```
pour trouver une paire d'itérateurs sur les éléments constituant la médiane d'une étendue d'itérateurs [first, last) dont les éléments ne sont pas triés. L'idée consistera à réutiliser l'algorithme `median` dans `median2` sur une copie locale triée de [first, last) pour en extraire les itérateurs sur la médiane, puis de rechercher les valeurs de ces itérateurs dans le [first, last) initial non trié.

4.5. Utilisez cet algorithme pour trouver la paire d'itérateurs de la médiane de `v`, puis affichez le ou les éléments constituant la médiane, et le cas échéant sa valeur. Dans la mesure où l'on n'utilisera pas la paire d'itérateurs résultante pour modifier `v`, on se contentera de **`const_iterator`**(s) lors de l'appel de `median2`.

4.6. Vérifiez la propriété $\frac{|\tilde{x}-\bar{x}|}{\sigma} \leq \sqrt{\frac{3}{5}}$ avec $\tilde{x}$: médiane, $\bar{x}$: moyenne & σ : écart type sur `v` ou `vt`.

5. Soustraction de la moyenne de v

Soustrayez à chaque élément de `v` la moyenne de `v` en utilisant uniquement des algorithmes et/ou des foncteurs standards. Puis affichez le `v` modifié.

6. Extraction des éléments positifs et négatifs de v

Copiez les éléments positifs de `v` dans `vp` et les éléments négatifs de `v` dans `vn` en utilisant uniquement des algorithmes et/ou des foncteurs standards. Puis affichez `vp` et `vn`.
Attention : `vp` et `vn` sont vides au départ, il faudra donc utiliser des « inserters » pour ajouter des éléments à la fin de `vp` et `vn`.

7. Changement de nature de conteneur

7.1. Peut on remplacer **`vector`** par **`deque`** dans la définition du type de conteneur utilisé au début de la fonction main ? Développez votre réponse en expliquant pourquoi.

7.2. Peut on remplacer **`vector`** par **`list`** dans la définition du type de conteneur utilisé au début de la fonction main ? Développez votre réponse en expliquant pourquoi.

8. Changement de nature de contenu

Quelles sont les opérations nécessaires à une classe X pour qu'elle puisse remplacer double dans la définition du type d'éléments `NumType` ?

9. Dépôt

A la fin de la séance, vous pourrez déposer une archive de votre code sur exam.ensiie.fr dans le dépôt **laoa24-tp1**. Vous pourrez utiliser la cible « archive » de votre projet pour régénérer une archive zip de votre code : `TP_STL-<date>.zip` qui sera placée dans le répertoire « `archives` » du projet.

Annexes

Types internes

Dans un algorithme (au sens de la STL), c'est à dire dans une fonction dont les arguments sont des itérateurs, il est parfois nécessaire de connaître la nature des types utilisés par ces itérateurs : la nature des valeurs, de la différence entre deux itérateurs, etc.

Pour ce faire on peut utiliser les `iterator_traits` relatifs aux itérateurs utilisés.

Par exemple :

```
template <typename Iterator>
Iterator algo(const Iterator & first, const Iterator & last)
{
    using value_t = typename iterator_traits<Iterator>::value_type;
    using diff_t = typename iterator_traits<Iterator>::difference_type;
    …
    diff_t nbElts = last - first;
    …
}
```

Si le type d'itérateur utilisé dans le programme utilisant cet `algo` n'est pas un type standard reconnu par les `iterator_traits`, cela provoquera une erreur de compilation.

Outils nécessaires

Pour réaliser ce TP vous aurez besoin :

- D'un compilateur C++ (g++, clang++, …)
- De CMake (evt CMake-GUI) qui pourra vous générer soit
  - Un Makefile pour compiler avec g++ ou clang++
  - Un projet/solution Visual Studio
- D'un IDE. Par exemple :
  - QtCreator : IDE par défaut de Qt, supporte les projets CMake.
  - Visual Studio Code (avec les plugins C/C++ et CMake).
  - CLion : IDE de JetBrains, supporte nativement les projets CMake.
  - Visual Studio (Community Edition) : devrait (?) supporter les projets CMake, mais il est sans doute plus simple de générer une solution Visual Studio avec CMake.

Importation des sources dans votre IDE

QtCreator (avec un CMakeLists.txt pour faire un projet CMake)
Importation des sources dans QtCreator

- File → Open File or Project …
- Naviguez jusqu'au fichier CMakeLists.txt et sélectionnez-le.
- Une page de configuration du projet s'ouvre
  - Sélectionnez uniquement le « Kit » commençant par Desktop Qt 5.xx.xx (qui correspond au Kit autodétecté) et déroulez « Details », pour spécifier le répertoire de build pour toutes les configurations Debug, Release, etc. : Créez un répertoire « build » à l'endroit où se trouvent vos sources et affectez-le à toutes les configurations de build.
  - Vous pouvez maintenant cliquer sur « Configure Project »

Visual Studio Code (VSC) (avec un CMakeLists.txt pour faire un projet CMake)

- Placez-vous dans le répertoire à la racine du projet et lancez la commande « `code .` » pour lancer VSC sur le répertoire courant. Si vous avez installé les plugins C/C++ et CMake dans VSC, il devrait reconnaître un projet cmake.