CIR2 - Algorithmique Avancée

Sujet TD&P n°3 - Séances 3 & 4

STL all-star game.

Algo & C++.

Dans cet exercice, il est question d'utiliser les conteneurs principaux de la **STL** en les faisant travailler en deux équipes : les conteneurs associatifs et les non-associatifs. L'équipe avec le meilleur chrono aura gagné.

Partie 1 - Génération de nombres premiers

i. (**Algo**). La première étape consiste à écrire un algorithme naïf qui vérifie si un nombre est un nombre premier, c'est-à-dire un entier naturel qui admet exactement deux diviseurs distincts entiers et positifs : 1 et lui même[1]. L'idée principale qui vous mènera à l'algorithme vérifiant qu'un nombre est premier est la suivante :

- **un nombre n est premier** si les restes (`modulo` noté `%`) de toutes les divisions entières de n possibles par des entiers inférieurs à n et supérieurs à 1, sont non nuls.
- Réfléchir ensuite à comment limiter le nombre d'itérations dans cette recherche de nombres premiers.

Partie 2 - Nombres premiers et C++

ii. (**C++**). **Générer tous les nombres premiers entre 1 et n**.

- Déclarer un entier constant n en lui donnant une valeur supérieure à 1000.
- Développer la fonction écrite en (**i.**).
- Écrire une nouvelle fonction qui appelle n fois la précédente et qui trouve tous les nombres premiers inférieurs à n.
- Sauvegarder ces nombres premiers dans un tableau `tabPremiers` de la STL `<array>`[2] :
  - https://en.cppreference.com/w/cpp/container/array,
  - https://www.geeksforgeeks.org/stdarray-in-cpp/ (ce deuxième lien devrait suffir et son utilisation est plus simple).

  `tabPremiers` doit être déclaré avec une taille inférieure à n mais reste assez grande pour tous les nombres premiers qui lui sont inférieurs. Par exemple, si `n==100`, il faut pouvoir accueillir les 25 nombres premiers.

---

[1] Les vingt-cinq **nombres premiers** inférieurs à 100 sont : 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, et 97.

[2] Le passage des `<array>` en paramètre étant relativement difficile, vous pouvez utiliser ici un `<vector>` à condition qu'il ne fasse pas partie de l'équipe des non-associatifs (questions iii.)

Samuel Deleplanque. (prénom.nom@yncrea.fr)

- Mettre en place un compteur `nbNbPremiers` qui représente le nombre d'éléments de `tabPremiers`.

Partie 3 - Déroulement de la compétition

iii. (**Lecture**). Dans la STL, on peut voir deux équipes de conteneurs : les associatifs (ils sont 4 : `set, multiset, map, multimap`) et les non-associatifs (`vector, list, queue, dequeue, stack`, heap etc.).

Dans le milieu associatif, les conteneurs se lassent des moqueries sur leurs piètres performances (e.g., l'association 'clé > valeur' prend plus de temps qu'un accès par indice). Ils organisent alors une compétition qui oppose les 4 associatifs à 4 autres conteneurs.

Les 2 équipes s'affrontent sur une course de relais où les nombres premiers sont échangés d'un conteneur à l'autre. En un tour, l'ensemble des nombres premiers doit passer par tous les conteneurs. La compétition se termine en 3 tours (cf. Figure 1). On fera attention à terminer chaque tour par le transfert des nombres premiers du dernier vers le premier conteneur (e.g., `vector` dans la la figure), ce dernier transfert représentant finalement le passage de la ligne d'arrivée lors du dernier tour.

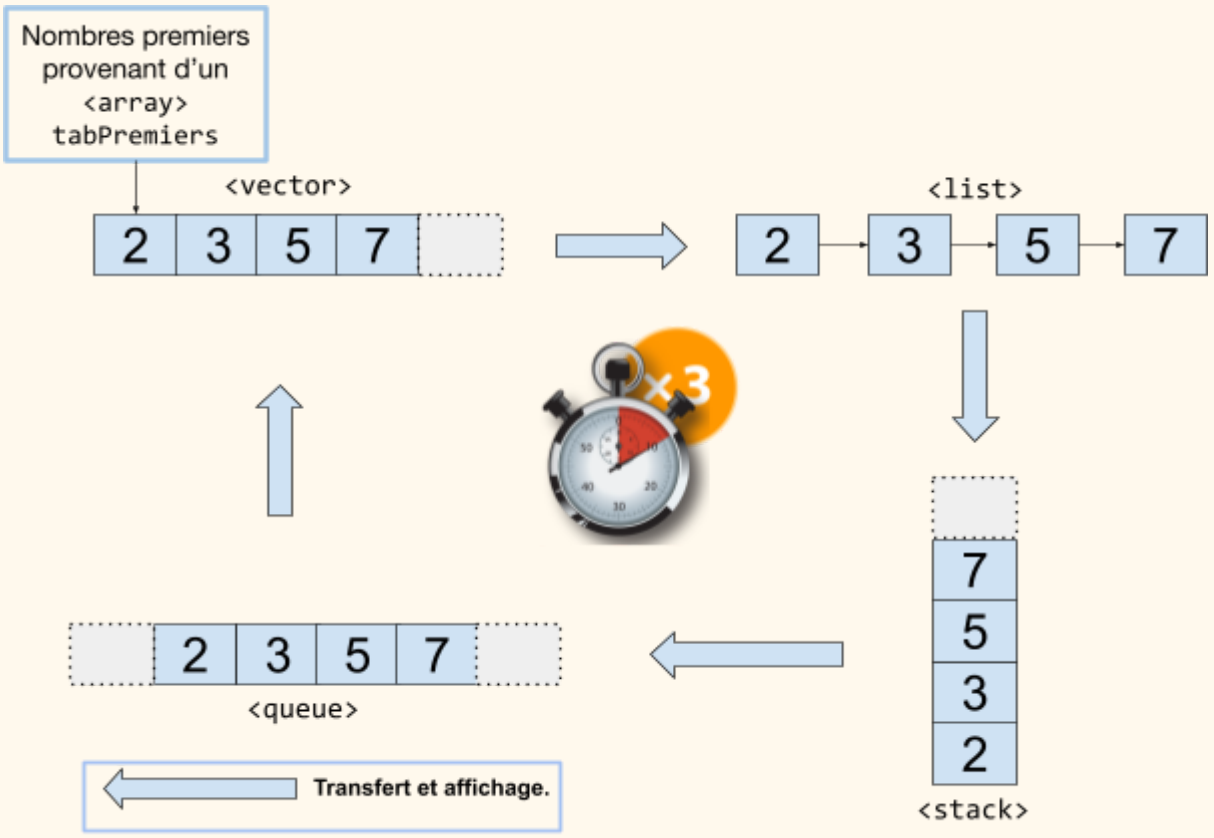

Figure 1. Exemple de parcours des nombres premiers dans l'équipe des non-associatifs. Avant de rentrer dans le "circuit" des quatre conteneurs, la génération des nombres premiers stocke ces derniers dans un `<array>`.

A vous de former l'équipe des "non-associatifs" dont la documentation de certains est donnée ici :

- stack https://fr.cppreference.com/w/cpp/container/stack
- liste https://fr.cppreference.com/w/cpp/container/list
- vecteur https://fr.cppreference.com/w/cpp/container/vector
- heap https://en.cppreference.com/w/cpp/algorithm/make_heap
- queue https://fr.cppreference.com/w/cpp/container/queue
- dequeue https://fr.cppreference.com/w/cpp/container/deque

L'équipe des associatifs est toute faite avec une instance de chacun de ces conteneurs :

- set https://fr.cppreference.com/w/cpp/container/set
- multiset https://fr.cppreference.com/w/cpp/container/multiset
- map https://fr.cppreference.com/w/cpp/container/map
- multimap https://fr.cppreference.com/w/cpp/container/multimap

Ces documentations vous renseignent sur les différentes opérations spécifiques à chaque conteneur (e.g., `pile.top(), liste.front()`...).

Partie 4 - Course de relais des nombres premiers

- **(C++) Implémenter le schéma expliqué ci-dessus**. Mettre en place deux boucles `for` bien distinctes (i.e., non imbriquées), une pour chaque équipe (associatifs et non-associatifs).
  Pour les `<map>` et les `<set>` on utilisera comme clés des entiers de 0 à `n-1`.

Prendre exemple sur le code C++ suivant où se passe le transfert des données entre un `vecteur` et une `liste` tout en les affichant[3] :

```
    while (vecteur.size()) {
        liste.push_back(vecteur.back());
        cout << " vector to list " << vecteur.back() << endl;
        vecteur.pop_back();
    }
```

[3] On notera ici que nous n'avons utilisé que des méthodes simples d'insertion et de suppression.

Remarque. Respectez les opérations principales de vos structures (cf. cours et documentation STL) avec par exemple, pour la pile (`std::stack`), un fonctionnement LIFO (Last In First Out) doit être respecté : tous les nombres premiers y sont alors insérés, puis ils ressortent de la pile dans un ordre inversé pour être donnés à la structure suivante. Selon vos facilités avec C++, vous pouvez tout à fait partir sur des conteneurs simples à manipuler jusqu'à même utiliser 4 instances de <vector> dans un premier temps à condition d'arriver à quatre conteneurs bien distincts pour chaque équipe.

Partie 5 - Déroulement de la compétition

iv. (**C++**). **Déterminer l'équipe gagnante en les chronométrant avec `chrono`**.

- Chercher une bonne documentation pour l'utilisation de chrono ou s'inspirer de :
  - *(https://www.geeksforgeeks.org/chrono-in-c/)*.
- Mettre en place l'affichage des temps réalisés par chaque équipe.
- Rendre le nombre de tours dépendant d'une constante, e.g. :
  - e.g., `constexpr int nbTours = 100;`
- Tenter de départager plus largement les deux équipes grâce à de grandes valeurs affectées à `nbTours`, ainsi qu'à n.

Partie Bonus

a. Afficher le contenu de chaque conteneur juste après qu'ils aient reçu la série de nombres premiers (et donc avant l'affichage lié aux transferts), et, pour cela, surcharger les opérateurs '<<' sur des conteneurs utilisés. Ce qui simplifiera grandement l'affichage de ces derniers. En voici deux exemples :

```
template <typename T>
ostream& operator<<(ostream& monStylo, const vector<T>& v)
{
  monStylo << "[";
  for (int i = 0; i < v.size(); ++i) {
      monStylo << v[i];
      if (i != v.size() - 1)
          monStylo << ", ";
  }
  monStylo << "]\n";
  return monStylo;
}
```

```
template <typename T, typename S>
ostream& operator<<(ostream& monCrayon, const map<T, S>& v)
{
  for (auto it : v)
      monCrayon << it.first << " : "
         << it.second << "\n";

  return monCrayon;
}
```

b. Améliorer l'affichage en manipulant `cout` :

- https://ideone.com/Yg8NKj (une barre de progression selon le nombre de tour),
- ou faire ses propres recherches sur la sortie `cout` et ses possibilités.

c. Grâce à un algorithme préparé par vos soin, et l'utilisation de plusieurs chronomètres, arrêter les championnats par équipe pour un sport individuel : établir un classement pour chaque conteneur sur les trois compétitions suivantes :

- les insertions les plus rapides,
- les accès les plus rapides,
- les suppressions les plus rapides.

Ici, pour chacune de ces 3 opérations, déterminer celle qui est la plus rapide pour chacun des conteneurs. Il n'est donc pas question ici d'ordonner les valeurs.

d. Établir le même protocole qui cette fois permettra de classer les conteneurs selon l'efficacité de leur algorithme `std::sort`. Autrement dit : répondre à la question suivant :

- Quel conteneur choisir pour un système qui requiert de très nombreux tris des données.

e. Reprendre l'ensemble du TP, bonus compris, en prenant des objets complexes à la place des nombres premiers. Vérifier que l'on obtient les mêmes classements.

Bon courage.