Série 7: Tableaux dynamiques avec `vector`

Lien avec le MOOC Initiation à la Programmation (en C++)
Lien avec ICC-Théorie en complément du MOOC

La suite de la partie théorique offre un peu moins d'opportunités d'illustrations en C++.

Cela est compensé par le lancement du projet cette semaine. A ce propos, le complément orienté projet N°3 porte sur les bonnes et les mauvaises pratiques pour aborder le projet.

Exercices semaine5 du MOOC

- **Document Tutoriel « moyenne de classe »**
  - Remplissage d'un tableau au fur et à mesure de la lecture des données car on ne connait pas a priori la taille qu'aura le tableau dynamique.

- **Document Exercices semaine 5 du MOOC**

  - **Exercice 15 : échauffement avec des tableaux dynamiques (niveau 1)**
    - Ajout d'un élément avec push_back

  - **Exercice 16 : produit scalaire (niveau 1)**
    - Passage par valeur de 2 tableaux **`vector`** à une fonction

  - **Exercice 17 : multiplication de matrices (niveau 2)**
    - Manipulation de tableaux **`vector`** à deux indices
    - Passage par référence de 2 tableaux à une fonction qui renvoie la valeur d'un tableau **vector**
    - Les arcanes de la *move semantics* ne sont pas au programme de notre cours

- **Document Exercices additionnels semaine 5 du MOOC**

  - **Exercice 14 : crible d'Erathostène (niveau 2)**
    - Cet algorithme a été mis au point il y a environ 2300 ans. La visualisation de son fonctionnement est très bien faite sur wikipedia. On peut aussi bien utiliser un **`vector`** ou un **`array`** de booléens car la taille est connue à l'écriture.

- **Document Tableaux « à la C »**
  - Lire ce document complémentaire avec les éléments de la classe inversée pour se faire une idée de l'organisation des données en mémoire. En particulier, dans le cas d'un tableau à deux indices l'organisation « ligne par ligne » permet de comprendre le coût constant de l'accès à un élément d'un tel tableau.

Complément Projet (3)

Vous disposez maintenant de la donnée, de la démo, d'un fichier source de départ et de fichiers de tests.

Par quoi commencer ? Comment s'organiser ?

Bonnes pratiques :

- d'abord prendre le temps de lire la donnée *tout en exécutant le programme de démo* sur les fichiers tests fournis.

- N'hésitez pas à commencer à *créer vos propres fichiers de tests* simplement en les écrivant avec geany.
    - Le programme de démo devrait répondre à la plupart de vos questions sur le comportement attendu pour toutes sortes de scénarios.

- **Tenez un journal** documentant l'avancement du projet, les questions en suspens, les choix effectués, les tests effectués, les bugs rencontrés. Ce document sera très utile dans vos interactions avec les assistants en cas de blocage.

- Continuez à faire les séries ; elles apportent une expérience précieuse et parfois même des éléments directement ré-utilisables dans le projet.

Mauvaise pratique : commencer à coder tête baissée.

Etapes conseillées :

- identifier la nature des données manipulées et comment elles sont organisées. Etant donné l'avancement du cours, une approche exploitant des **vector** est suffisante et sera acceptée pour le rendu final.

- identifier les **actions** sur ces données, soit selon la stratégie top-down ou bottom-up.

- dans votre journal, dessiner une vue d'ensemble de la résolution en délégant les sous-problèmes à des fonctions jusqu'à atteindre un niveau suffisamment simple.

> Remarque : dans le programme final la taille d'une fonction ne devra pas dépasser une taille écran (40 lignes) et une ligne ne doit pas dépasser 87 caractères.

- effectuer la mise au point fonction par fonction, en validant chaque fonction avec un petit programme de test, avant de passer à la suivante.

> C'est une erreur grossière d'écrire tout le code source d'un seul coup puis d'appuyer sur le bouton « build »…

- Vous pouvez commencer à tester la fonction main() en utilisant des fonctions vides ou très simplifiées (appelées « *stub* ») pour les fonctions traitant les sous-problèmes et qui ne sont pas encore finalisées. Cela permet de valider certains aspects comme la lecture des données et la transmission de données entre les différents niveaux de fonctions.