MORAIS Philippe **Grenoble INP - Ensimag**
Zhu Joël **Mai 2020**

Modélisation et Programmation en C++

Génération de nombres aléatoires

TP 1

Question 1

`Dvector x; x = Dvector(3, 1.);`

Le compilateur fait d'abord appel au constructeur par défaut pour initialiser la variable. Puis il fait un appel au constructeur défini par l'utilisateur et réalise une affectation grâce à la surcharge de l'opérateur = .

`Dvector x = Dvector(3, 1.);`

Le compilateur fait ici directement appel au constructeur défini par l'utilisateur.

TP 2

Question 2

On compare les deux opérateurs externes suivant :

`Dvector operator+(Dvector a, Dvector b)`

Une copie de chacun des paramètres est créée à l'appel (implicite) de cette fonction.

`Dvector operator+(const Dvector &a, const Dvector &b)`

On passe ici l'adresse en paramètre de la fonction. Ainsi, on accède directement à l'objet sans le recopier. On indique const pour indiquer qu'on souhaite seulement accorder le droit à l'accès à l'objet, mais pas la modification.

Dans le 1er cas, on crée deux objets supplémentaires alors que dans le deuxième cas, aucun objet supplémentaire n'est créé.

Question 4

On peut identifier d'autres factorisations possibles comme le lien entre les opérateurs de type += et + , ou encore que l'opérateur != n'est que la négation de == .

Question 5

On compare les performances de l'opérateur = si on utilise ou non la fonction memcpy pour recopier les données sur des Dvector de 50 millions d'éléments. En particulier, on mesure le temps nécessaire à la recopie sans utiliser memcpy et lorsque l'opérateur ( ) vérifie le non dépassement des bornes. Valeurs moyennes sur cinq mesures consécutives exprimées en millisecondes :

Performance de =	Avec vérification des bornes	Sans vérification des bornes
Avec memcpy	1 776 ms	1 834 ms
Sans memcpy	2 573 ms	2 418 ms

On remarque que la vérification des bornes n'a pas un impact significatif sur les performances (moins de 1 %). Par contre, l'utilisation de memcpy réduit le temps d'exécution de cet opérateur d'environ 30 %.

TP 3

Question 2

On évalue les performances temporelles des générateurs implémentés pour générer une suite de 10 millions de nombres pseudo-aléatoire. Valeurs moyennes sur cinq mesures consécutives exprimées en millisecondes :

Générateur	Temps
Park-Miller standard (X_0 = 17)	1 354 ms
Park-Miller standard (X_0 = 23671)	1 036 ms
Xorshift ($a_1 = 21,\ a_2 = 35\ ,\ a_3 = 4,\ X_0 = 1$)	1 021 ms
Xorshift ($a_1 = 17,\ a_2 = 31\ ,\ a_3 = 8,\ X_0 = 1$)	988 ms

Le générateur Xorshift semble être en moyenne légèrement plus rapide que la méthode de Park-Miller.

Question 3

Un moyen de mettre en évidence les forces et faiblesses des générateurs aléatoires implémentés serait de comparer pour une suite de nombres générés les moyennes et variances empiriques avec leurs valeurs théoriques. Plus généralement, on peut réaliser des tests statistiques d'adéquation pour savoir si le vecteur généré suit bien la loi attendu comme le test de Kolmogorov-Smirnov.