Modélisation et programmation, TP2

Felix HAHNLEIN et Auriane REVERDELL

Février 2016

1 Réponses aux questions de la partie Analyse

1.1 Question 2

On va d'abord s'intéresser à la première écriture:

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

Ici les arguments sont des `Dvector`. Ces variables passées en argument vont être copiées en variables locales, les modifications des variables locales dans la fonction appelée ne vont pas modifier les variables initiales, elles ne s'appliquent qu'à la copie.

On s'intéresse ensuite à la deuxième écriture:

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

Ici les paramètres sont des références sur des objets de type `Dvector`. L'utilisation de la référence est utile ici car elle prend une copie de l'objet tout en ayant la même adresse que ce dernier pour pouvoir le modifier. Dans cette méthode on veut seulement pouvoir lire les deux objets sans pouvoir les modifier (d'où l'ajout du `const`). En effet cela donne une sécurité supplémentaire : on ne veut pas modifier les variables de la fonction appelante par la fonction appelée.

Finalement dans le 1^{er} cas deux objets (les copies) sont créés puis détruits en sortant de la fonction. Et dans le 2^{me} cas, aucun objet n'est créé.

1.2 Question 4

Nous avons décidé de privilégier une meilleure maintenabilité par rapport à de meilleures performances. Voilà les opérations réutilisées :

- Utilisation de la fonction accesseur `[]` pour un accès en lecture et en écriture dans toutes les fonctions en ayant besoin (du TP1 et du TP2).
- Utilisation de l'opérateur `+` avec un décimal à droite pour l'opérateur `+` avec un décimal à gauche (commutativité).
- Utilisation des opérateur `*` pour multiplier par `-1` et `+` pour faire l'opérateur `-`.
- Utilisation de l'opérateur `*` à droite pour faire celui à gauche (commutativité).
- Utilisation de l'opérateur `*` pour faire l'opérateur `/`.
- Utilisation de l'opérateur `*` pour faire l'opérateur `/`.
- Utilisation de l'opérateur `*` pour faire l'opérateur `-` unaire.

- Utilisation de l'opérateur `+` entre deux vecteurs pour faire l'opérateur `-` entre ces deux vecteurs.
- Utilisation de `+=` pour l'implémentation de `-=`.
- Utilisation de `*=` pour l'implémentation de `/=` décimal.
- Utilisation de `==` pour l'implémentation de `!=`.

1.3 Question 5

Comparaison des performances de l'opérateur `=` :

Nous avons exécuté le programme de test `test_affectation_three`, qui recopie 2 millions de valeurs, dix fois sur la machine suivante:

```
$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                4
On-line CPU(s) list:   0-3
Thread(s) par coeur :  1
Coeur(s) par support CPU :4
Socket(s):             1
Noeud(s) NUMA :        1
ID du vendeur :        GenuineIntel
Famille CPU :          6
Modèle :              60
Version :              3
CPU MHz :              800.000
BogoMIPS:              6584.60
Virtualisation :       VT-x
L1d cache :            32K
L1i cache :            32K
L2 cache :             256K
L3 cache :             6144K
NUMA node0 CPU(s):     0-3
```

Voici les moyennes obtenues pour des vecteurs de taille 1 000 000 000 :

sans memcpy	avec memcpy
24.276 s	19.343 s

Nous constatons que la recopie manuelle des données est moins efficace que l'utilisation de la fonction `memcpy`. En effet, il s'agit d'une fonction optimisée pour la copie de données, alors que notre parcours séquentiel va faire apparaître des instructions assembleurs superflues.