Algorithmique et structures de données 2

X4I0030 **2015 /2016**

Projet 1 [3 séances]

Objectifs

1. Grand angle

Votre travail consiste à comparer des implémentations de la structure d'Anneau générique décrite dans le sujet de Travaux Pratiques n° 1. Cette comparaison porte avant tout sur le coût spatial et le coût temporel, mais vous pouvez argumenter sur d'autres qualités du code (utilisabilité, fiabilité, maintenabilité, lisibilité, cf par exemple http://rmod.lille.inria.fr/archives/papers/Mord14a-Chapitrequalite.pdf).

Rappel : les méthodes de la classe Anneau sont estVide, ajoute, supprime, courant, avance, recule.

2. Zoom

Deux implémentations sont imposées, mais vous pourrez en proposer une troisième de votre choix (par exemple celle du TP1) :

a) La première, une classe Anneau possédant une file en attribut (et peut-être quelques autres informations de type simple). Pour la file, utilisez la structure disponible dans la STL (explication par exemple ici : http://www.cplusplus.com/reference/queue/queue/). Le patron de classe queue s'instancie avec un conteneur à choisir parmi deque et list (vous pouvez choisir celui que vous voulez, deque est choisi en l'absence d'indication).

```
 // Exemple d'utilisation d'une file (queue) de la STL
#include <iostream>       // cout, endl
#include <queue>          // queue

int main ()
{
  std::queue<int> uneFile;                    // basée sur deque
  if (uneFile.empty()) { std::cout << "vide"<< std::endl;} // vide ?
  uneFile.push(7);
  uneFile.push(13);                           // enfile
  std::cout << uneFile.front() << std::endl;  // premier
  uneFile.pop();                              // defile
  std::cout << uneFile.front() << std::endl;
  return 0;
}
```

b) Deuxièmeme, une classe Anneau possédant deux piles en attribut (et peut-être quelques autres informations de type simple). L'élément courant sera le sommet de l'une des piles, toujours la même. Pour les piles, utilisez la structure disponible dans la STL (explication par exemple ici : http://www.cplusplus.com/reference/stack/stack/). Le patron de classe stack s'instancie avec un conteneur à choisir parmi deque et list (vous pouvez choisir celui que vous voulez, deque est choisi en l'absence d'indication).

```
 // Exemple d'utilisation d'une pile (stack) de la STL
#include <iostream>       // cout, endl
#include <stack>          // stack

int main ()
{
  std::stack<int> laPile;                    // basée sur deque
  if (laPile.empty()) { std::cout << "vide"<< std::endl;} // vide ?
  laPile.push(7);
  laPile.push(13);                          // empile
  std::cout << laPile.top() << std::endl;   // sommet
  laPile.pop();                             // depile
  std::cout << laPile.top() << std::endl;
  return 0;
}
```

3. Mise au point

Vous rédigerez en parallèle de la première implémentation un programme de test permettant de vérifier les méthodes au fur et à mesure que vous les écrivez.

Ce programme doit pouvoir être réutilisé pour vérifier l'autre implémentation, sans modification (sauf éventuellement un #include).

Chambre noire

4. Focus

Une fonction utilisatrice `dedoublonne` servira de témoin pour la comparaison des implémentations. Il s'agit d'une fonction prenant en paramètre un Anneau A et retournant un Anneau ayant chacun des éléments de A en un seul exemplaire.

Cette fonction doit être écrite en dehors de la classe Anneau (hors spécification) et n'utilisera donc que les méthodes publiques décrites en 1). Elle n'utilisera pas d'autre structure de données que l'anneau passé en paramètre et l'anneau qui sera renvoyé.

Il faudra bien entendu écrire un programme de test pour vérifier sa correction.

5. Développement

Vous devez en outre écrire un programme de mesure contenant la fonction `dedoublonne` et l'appelant sur des anneaux dont le contenu est choisi aléatoirement.

Ce programme doit mesurer le temps (en millisecondes) d'exécution de la fonction pour chaque appel et afficher un tableau des temps de calcul en fonction du nombre de valeurs stockées dans l'anneau ; il s'agit ici à chaque fois d'un temps moyen sur un échantillon d'annneaux de même taille.

Ce programme doit pouvoir fonctionner avec chaque implémentation sans modification du code (à l'exception éventuelle d'un `#include`).

6. Tirage

Un document de travail doit être rédigé au préalable et **en parallèle**. Ce document présentera les choix effectués (constantes, comportement des méthodes non précisé dans l'énoncé, choix d'implémentation – élément courant en queue ou en tête, nombre d'éléments stocké, ...) et les limites d'utilisation. Il deviendra le rapport de programmation et est aussi, si ce n'est plus important, que le code.

Il devra comporter un comparatif de vos implémentations en terme de complexité temporelle (l'ordre de grandeur du coût de chaque méthode doit être renseigné, s'appuyant sur les complexités des structures piles et files utilisées) et d'encombrement mémoire.

L'algorithme de la fonction `dedoublonne` y sera expliqué et son cout temporel estimé.

Ce rapport comportera aussi la description de la méthode de mesure utilisée pour étudier `dedoublonne` et un graphique comparant les temps obtenus en fonction du nombre d'éléments pour chaque implémentation. Discutez les résultats obtenus.

Rendu

Votre travail est à déposer sur Madoc avant le 13 mars 2016 à 13h03 sous forme d'une archive zip contenant en particulier

a) le rapport au format pdf (une dizaine de pages au maximum hors annexes) ,

b) le code source des implémentations (un dossier par implémentation),

c) le code source du programme de test des implémentations,

d) le code source de la fonction `dedoublonne` et du programme la vérifiant,

e) le code source du programme de mesure de la fonction `dedoublonne`.

Le code doit être clair et concis, commenté, avec en particulier les pré- et post-conditions.

Le rapport doit être bien écrit (français correct, attention à l'orthographe!) et agréable à lire, il doit comporter introduction et conclusion et éventuellement des annexes pour des détails techniques (graphiques, code, etc.)