Université du Luxembourg 2004–2005

D.U.T. Informatique 2ème année mercredi 3 novembre

Programmation avancée en langage C++
Examen oral

`Sebastien.Varrette@imag.fr` (Bureau BR.4.07)

Durée: 30 minutes

Tout document interdit. Les exercices suivants sont à réaliser sur machine et devront être ensuite présentés et justifiés aux correcteurs.

Gestion de Piles

1. *Ecriture de classes*

   Ecrire une classe `Stack` représentant une pile FIFO [1] d'entiers ayant les caractéristiques suivantes :
   - la pile devra reposer sur un tableau de dimension n, n étant passé en paramètre au constructeur (allocation dynamique)
   - la classe `Stack` fournit un constructeur par recopie ainsi qu'un destructeur.
   - elle permet d'empiler (méthode `push(int)`) et de dépiler (méthode `pop()`) des données de type `int`.
   - une fonction `length()` permet de savoir combien de données sont empilées.
   - une fonction `size()` permet de connaitre la taille de la pile.
   - une fonction `print()` permet d'afficher l'état de la pile.

   La figure 1 illustre l'ajout successif de valeurs suivie d'une suppression-dans une pile de taille $n = 3$.

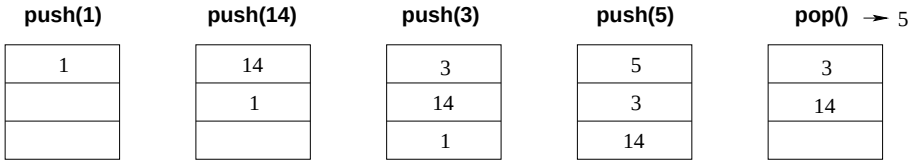

Figure 1 – ajout successif puis suppression de valeurs dans une pile de taille $n = 3$

2. *Surcharge d'opérateurs*
   - Surcharger l'opérateur `==` permettant de savoir si deux piles sont identiques.
   - Surcharger l'opérateur `()` afin d'accéder directement (et eventuellement affecter) la valeur en tête de pile (sans la dépiler).

   Ainsi, si `s` est la pile représentée à l'issue de la figure 1, on doit pouvoir considérer la séquence d'instructions suivantes :

```
Stack s1 = s;
```

1. First In, First Out

```
s.print()                 // affiche "[nb Elem : 2] 3 -> 14"
cout << s() << endl;      // affiche "3"
s.print()                 // affiche "[nb Elem : 2] 3 -> 14"
s() = 10;                 // affectation de la valeur de tête
s.print()                 // affiche "[nb Elem : 2] 10 -> 14"
cout << s == s1 << endl; // affiche "0"
```

3. *Héritage*

   Définir la classe `PriorityStack` qui hérite de la classe `Stack` en ajoutant un paramètre entier caractérisant une valeur de priorité de la pile. Illustrer l'utilisation de cette classe en faisant intervenir les différentes méthodes qui la compose.

4. *Utilisation de modèle*

   Reprendre la définition de la classe `Stack` en utilisant cette fois-ci les `template` pour permettre de gérer des éléments de n'importe quel type. On définira ainsi la classe `TStack` et on illustrera son utilisation sur une pile de caractères et de flottants.