Université de Paris VI **M1 - IRSI - I4 - CIMES**

18 mars 2019

documents autorisés : cours, TDs, TPs
Ordinateurs, tablettes, smartphones non autorisés
durée : 1h30

Le langage C++ ne dispose pas de type de variable *compteur*, raison pour laquelle on utilise le type entier *int* pour réaliser des opérations de comptage. Créer un type `Counter` permettrait de disposer d'un type de variable uniquement dédié à des opérations de comptage. On propose donc dans cet examen de mettre au point une classe `Counter` pour déclarer des objets se caractérisant par :

- une valeur entière, positive ou nulle et valant 0 à l'origine (à la création de l'objet) ;
- deux opérations d'incrémentation et de décrémentation de pas quelconque ;
- la décrémentation d'un compteur valant déjà 0 est sans effet.

La classe comportera un unique attribut `private: int value;` permettant de conserver la valeur courante du compteur.

Question 1 : initialisation Proposer la définition de la classe `Counter` permettant de réaliser les opérations suivantes :

```
int main() {
  Counter i1; // declaration d'une variable de type Counter
  Counter i2(10); // declaration d'un compteur initialise a la valeur 10
  Counter i3 = i1; // declaration d'un compteur initialise a partir du compteur i1
}
```

Question 2 : convertion Ecrire le constructeur qui permet de convertir un entier en un compteur comme dans :

```
int main() {
  Counter i1;
  int var;

  i1 = Counter(var);
}
```

Question 3 : incrémentation/décrémentation. Compléter la définition de la classe `Counter` afin de pouvoir réaliser les opérations suivantes :

```
int main() {
  Counter i1, i2, i3;

  i1.inc(); // incrementation de 1 pas du compteur i1
  i2.dec(); // decrementation de 1 pas du compteur i2
  i3.dec(10); // decrementation de 10 pas du compteur i3
}
```

Question 4 : affichage du compteur On souhaite pouvoir afficher directement la valeur du compteur à l'écran. Proposer une surcharge de l'opérateur d'insertion de flux permettant l'opération suivante :

```
int main() {
  Counter i1=10;

  cout << "La valeur du compteur est: " << i1 << endl;
}
```

Question 5 : affectation. Afin de pouvoir affecter des valeurs entières à des compteurs, on demande de surcharger l'opérateur d'affectation = de sorte à rendre possible les opérations suivantes :

```
int main() {
  Counter i1, i2, i3;

  i1 = 10;
  i2 = i3 = 20;
}
```

Question 6 : comparaison. Proposer une surcharge de l'opérateur de comparaison < entre deux compteurs. Le résultat doit être un booléen et permettre des commandes comme :

```
int main() {
  Counter i1=10, i2=20
  if( i1 < i2) {
     ...;
  }
}
```

Question 7 : comparaison. Le compilateur accepterait-il une boucle écrite comme ci-dessous (sachant les méthodes et opérateurs définis jusqu'à présent) ? Expliquer la réponse.

```
int main() {
  for( Counter i=0; i<10; i.inc() ) {
     ...
  }
}
```

Question 8 : incrémentation++. Reprendre la question 3 et proposer une surcharge de l'opérateur `operator++()` de sorte à pouvoir écrire :

```
int main() {
  for( Counter i=0; i<10; i++ ) {
     ...
  }
}
```

Attention que l'écriture suivante : `i1 = i2++` doit également être possible, c'est à dire l'affectation suivie d'une incrémentation (post-fixée), opération qui doit donner le même résultat que : `i1 = i2; i2++;`.

Question 9 : affectation élargie. Reprendre la question 4 (affectation) et proposer les deux opérateurs d'affectation élargie `operator+=()` et `operator-=()`. Ces surcharges doivent permettre l'incrémentation/décrémentation à un pas quelconque :

```
int main() {
  Counter i1=0;

  i1 += 10;
  i1 -= 2;
  }
}
```