Correction examen de Programmation Modulaire

—Licence MI/IM - Info 3—

▸ **Exercice 1. (Programmes faux)** – sur 3 points –

- Programme 1 : l'attribut `i_` de la classe `C` est privé. Il n'est pas accessible dans l'opérateur d'affichage. Pour corriger, remplacer la ligne 14 par

```
    out << t.get_i();
```

  Alternative : déclarer l'opérateur ami de la classe. Ajouter après la ligne 9 :

```
    friend ostream &operator<<(ostream, const C &);
```

- Programme 2. La classe ne déclare pas de constructeur pas défaut qui est appelé par la ligne 14. Correction : remplacer les lignes 14 et 15 par

```
    C t{4};    // alternative C t(4);
```

  Alternative : la solution qui consiste à déclarer un constructeur par défaut change le comportement de la classe. Elle permettrait à l'utilisateur de créer un objet sans donner de valeur explicite à l'attribut. Elle est moins bonne. Elle consiste à ajouter après la ligne 6 :

```
    C() : i_{0} {}   // Pourquoi 0 plutôt qu'un autre nombre ???
```

  Note : il est faux de dire que l'on a seulement déclaré la variable mais qu'on ne l'a pas définie. Elle est définie au sens que l'emplacement mémoire est réservé, mais l'initialisation ne peut avoir lieu faute de constructeur.

Problème : Réseau de Bus

▸ **Exercice 2. (Horaires)** – sur 7.5 points –

1. Voici le fichier `horaire.hpp`

```
#ifndef HORAIRE_HPP
#define HORAIRE_HPP

#include <iostream>
using namespace std;

class Horaire {
  int heures;          /* valeur comprise entre 0 et 23 inclus */
  int minutes;        /* valeur comprise entre 0 et 59 inclus*/

 public:

  Horaire(int h, int m);
  bool operator==(Horaire other) const;
  bool operator<(Horaire other) const;
  void imprime(ostream & out) const;  // Variante valeur de retour ostream &
};

ostream & operator<<(ostream & out, Horaire t);

#endif
```

2. Définition du constructeur de la classe `Horaire`.

```
Horaire::Horaire(int h, int m) : heures{h}, minutes{m} {
  if (minutes < 0 or minutes >= 60 or heures < 0 or heures > 23)
    throw invalid_argument("Heure invalide");
}
```

3. Dans quel fichier doit-on écrire cette définition ? Dans le fichier `horaire.cpp`
4. Écrire la définition de l'opérateur de comparaison <.

```
bool Horaire::operator<(Horaire other) const {
  return heures < other.heures
      or (heures == other.heures and minutes < other.minutes);
}
```

5. Écrire la définition de la méthode `imprime`.

```
void Horaire::imprime(ostream & out) const {
  out << setw(2) << setfill('0') << heures;
  out << ":";
  out << setw(2) << setfill('0') << minutes;
}
```

6. En utilisant la méthode `imprime`, surcharger l'opérateur d'affichage pour les `Horaire`.

```
ostream & operator<<(ostream & out, Horaire t) {
  t.imprime(out);
```

```
  return out;
}
```

7. Donner les deux lignes d'un fichier `Makefile` permettant de compiler un fichier `horaire.o` ; la première ligne devra donner toutes les dépendances et la deuxième ligne la commande de compilation.

```
horaire.o : horaire.cpp horaire.hpp
⟶g++ -std=c++11 -Wall horaire.cpp -c
```

Alternatives à la commande de compilation (l'ordre n'importe pas, et on peut ajouter des options d'optimisation ou de debug) :

```
⟶g++ -c -std=c++11 -Wall -g -O3 horaire.cpp
```

ou encore, en utilisant les variables standards :

```
⟶$(CXX) -c $(CXXFLAGS) horaire.cpp
```

▸ **Exercice 3. (Lignes de bus)** – 8 points –

1. Donner les directives d'inclusion

```
Fichier horaire.hpp:
   rien
Fichier horaire.cpp:
   #include "horaire.hpp"
Fichier arret.hpp:
   #include "horaire.hpp"
Fichier arret.cpp:
   #include "horaire.hpp"    // Pas obligatoire, déjà inclus dans le .hpp
   #include "arret.hpp"
Fichier ligne.hpp:
   #include "horaire.hpp"    // Pas obligatoire, déjà inclus dans arret.hpp
   #include "arret.hpp"
Fichier ligne.cpp:
   #include "horaire.hpp"    // Pas obligatoire, déjà inclus dans ligne.hpp
   #include "arret.hpp"      // Pas obligatoire, déjà inclus dans ligne.hpp
   #include "ligne.hpp"
```

2. Pour chacune des deux classes `Arret` et `Ligne`, écrire la ligne du `Makefile` qui donne la liste des dépendances du fichier `.o` associé.

```
arret.o : arret.cpp arret.hpp horaire.hpp
ligne.o : ligne.cpp ligne.hpp arret.hpp horaire.hpp
```

   L'énoncé ne demande pas la commande

3. Déclarer et définir en une seule ligne le getter `get_nom` qui retourne le nom d'un `Arret`. Dans quel fichier et à quel endroit placez-vous cette déclaration ?

```
  string get_nom() const { return nom; };
```

   Dans la partie publique de la classe `Arret` du fichier `arret.hpp`

4. Déclaration de la méthode `prochain_passage`

```
  int prochain_passage(Horaire t) const;
```

5. Donner la définition de la méthode `prochain_passage` précédente.

```
int Arret::prochain_passage(Horaire t) const {
  for (int i=0; i < int(passages.size()); i++)
    if (t == passages[i] || t < passages[i]) return i;
  throw runtime_error("Plus de passages");
}
```

   Alternative : retourner -1 au lieu de lever une exception.

6. Déclaration et définition de la méthode `ajoute_passage` :

```
  void ajoute_passage(Horaire t);
```

```
void Arret::ajoute_passage(Horaire t) {
  if (passages.size() != 0 and passages.back() < t)
    passages.push_back(t);
  else throw invalid_argument("Pas près le dernier passage");
}
```

Ligne

7. Constructeur de la classe `Ligne`.

```
Ligne::Ligne(string s, vector<Arret> a) : nom{s}, arrets{a} {
  if (a.size() < 2)
    throw invalid_argument("La ligne doit avoir au moins deux arrets");
  for (int i=1; i < int(arrets.size()); i++) {
    if (arrets[i].nb_passages() != arrets[0].nb_passages())
      throw invalid_argument("Pas le meme nombre de passages");
  }
  for (int p = 0; p < arrets[0].nb_passages(); p++) {
    for (int i=0; i < int(arrets.size()) - 1; i++) {
      if (not (arrets[i].ieme_passage(p) < arrets[i + 1].ieme_passage(p)))
          throw invalid_argument("Le bus ne peut pas remonter le temps");
    }
  }
}
```

8. Donner la déclaration et la définition d'une méthode `cherche_arret` qui prend une chaîne de caractères et qui retourne la position de l'arrêt dont c'est le nom dans la ligne, ou -1 s'il n'y a pas d'arrêt portant ce nom.

```
  int cherche_arret(string a) const;
```

```
int Ligne::cherche_arret(string s) const {
  for (int i = 0; i < int(arrets.size()); i++)
    if (arrets[i].get_nom() == s) return i;
  return -1;
}
```

9. Méthode `voyage`

```
array<Horaire, 2> Ligne::voyage(string dep, Horaire t, string dest) const {
  int idep = cherche_arret(dep);
  int idest = cherche_arret(dest);
  if (idep == -1 or idest == -1 or idest <= idep)
    throw runtime_error("Trajet impossible sur cette ligne");
  int rang = arrets[idep].prochain_passage(t);
  return {{ arrets[idep].ieme_passage(rang), arrets[idest].ieme_passage(rang)}};
}
```

▸ **Exercice 4. (Réseau de Bus)** – 2 points –

1. Écrire une méthode de la classe `Reseau` qui calcule l'horaire d'arrivée d'un `Trajet` tout en vérifiant si le trajet est bien cohérent.

```
Horaire Reseau::horaire_arrivee(const Trajet &t) const {
  Horaire h = t.hDepart;
  string position = t.etapes[0].nomDepart;

  for (int nb = 0; nb < int(t.etapes.size()); nb += 1) {
    /* pour chaque etape:
     * 1. verifier si le point d'arrivee de l'etape precedente est bien le point
     * de depart de l'etape courante.
     * 2. calculer son heure d'arrivee (verifie si l'etape peut bien etre parcourue)
     */
    Etape e = t.etapes[nb];
    // Pour éviter la copie : const Etape &e = t.etapes[nb];
    if (position != e.nomDepart) {
      throw runtime_error("On ne repart pas d'ou on vient d'arriver");
    }
    /* horaire de fin de l'etape courante */
    h = lignes[cherche_ligne(e.nomLigne)].voyage(position, h, e.nomArrivee)[1];
    position =  e.nomArrivee;
  }
  return h;
}
```