université PARIS-SACLAY

Correction partiel de Programmation Modulaire

—Licence MI/IM/MNSI - Info 3—

Aucun document n'est autorisé à part la fiche résumé de *C++*, où vous pouviez consigner des notes manuscrites personnelles au verso. Tous les exercices sont indépendants. Même si l'on ne sait pas répondre à une question, on peut utiliser la réponse dans la suite de l'exercice. Une grande importance sera accordée à la qualité de la rédaction (lisibilité, indentation,...).
Le barème est indicatif et pourra changer à la correction.
Durée : 2h00.

▸ Exercice 1. (Question de cours) – sur 3 points –

Répondre en une ou deux phrases aux questions suivantes

1. Que veux dire «surcharger une fonction» ? Que fait le compilateur quand il voit une surcharge de fonction ? Donner un exemple.

✂ ----------

C'est quand deux fonctions différentes portent le même nom mais avec des types de paramètres différents. Le compilateur choisit la fonction dont le type des paramètres correspond.

Exemple : `abs` pour les `float` et les `int`.

---------- ✂

2. Qu'est ce qu'une «méthode» ? Comment appelle-t-on une méthode depuis le programme ? Donner un exemple.

✂ ----------

C'est une fonction qui est déclarée à l'intérieur d'une classe (fonction membre). On appelle une méthode sur un objet de la classe avec la notation pointée.

Exemple : la méthode `size` des vector. Appel avec `v.size()`.

---------- ✂

▸ Exercice 2. (Entiers signés) – sur 8 points –

On rappelle qu'en `C++` le type `unsigned` représente des entiers sans signe. Le but de cet exercice est de représenter les entiers relatifs (avec signe) sur une machine qui ne dispose que des entiers naturels (positifs). On s'interdit donc d'utiliser le type `int`, on utilisera seulement `unsigned`. Pour simplifier, on va supposer que la multiplication de deux `unsigned` donne toujours un résultat correct (pas de débordement de capacité).

1. On veut représenter les entiers relatifs par une structure comportant deux champs, l'un de type booléen pour le signe (positif est codé par `true`, et négatif par `false`) et l'autre champ de type `unsigned` pour la valeur absolue. Écrire la déclaration de la structure `Relatif` pour coder les entiers relatifs.

✂ ----------

```
9  struct Relatif {
10   bool signe;
11   unsigned vabs;
12 };
```

---------- ✂

2. Écrire une fonction `abs` qui retourne la valeur absolue d'un `Relatif`. Le résultat retourné sera de type `Relatif`.

✂ ----------

```
16 Relatif abs(Relatif a) {
17   return {true, a.vabs};
18 }
```

---------- ✂

3. Surcharger l'opérateur d'égalité pour les `Relatif`.

```
21 bool operator==(Relatif x, Relatif y) {
22   return x.signe == y.signe and x.vabs == y.vabs;
23 }
```

4. Proposer en utilisant l'infrastructure doctest un cas de test comportant plusieurs tests pour l'opérateur d'égalité. On fera attention à bien tester tous les comportements.

```
33 TEST_CASE("Fonction abs sur les Relatif") {
34   CHECK(Relatif{true, 3} == Relatif{true, 3});
35   CHECK(Relatif{false, 3} == Relatif{false, 3});
36   CHECK_FALSE(Relatif{true, 3} == Relatif{false, 3});
37   CHECK_FALSE(Relatif{false, 3} == Relatif{false, 4});
38 }
```

5. En utilisant l'égalité précédemment définie, proposer un cas de test comportant plusieurs tests pour la fonction `abs`.

```
41 TEST_CASE("Fonction abs sur les Relatif") {
42   CHECK(abs(Relatif{true, 3}) == Relatif{true, 3});
43   CHECK(abs(Relatif{false, 3}) == Relatif{true, 3});
44   CHECK(abs(Relatif{false, 5}) == Relatif{true, 5});
45   CHECK(abs(Relatif{true, 5}) == Relatif{true, 5});
46 }
```

6. Surcharger l'opérateur de multiplication pour les nombres `Relatif`.

```
50 Relatif operator*(Relatif a, Relatif b) {
51   return {a.signe == b.signe, a.vabs * b.vabs};
52 }
```

ou bien

```
1 Relatif operator*(Relatif a, Relatif b) {
2   Relatif res;
3   res.vabs = a.vabs * b.vabs;
4   res.signe = (a.signe == b.signe);
5   return res;
6 }
```

7. On veut maintenant que `Relatif` soit une classe, qui contienne un constructeur à partir d'un `unsigned`, ainsi qu'une méthode `abs` (correspondant la fonction `abs` précédente). Écrire la déclaration de la classe.

```
9  struct Relatif {
10   bool signe;
11   unsigned vabs;
16   Relatif(unsigned);
17   Relatif abs() const;
18 };
```

8. Écrire la définition du constructeur (en dehors de la classe).

```
27 Relatif::Relatif(unsigned x) : signe{true}, vabs{x} {}
```

9. Écrire la définition de la méthode `abs`.

```
22 Relatif Relatif::abs() const {
23   return {true, vabs};
24 }
```

▸ **Exercice 3. (Bibliothèque)** – sur 9 points –

Une bibliothèque municipale prête à ses adhérents au plus 10 livres (ou BDs ou journaux) par adhérent et ce, pour une durée de 3 semaines maximum. Les prêts sont gratuits mais les retours en retard sont payants : tout retard est facturé $0,20$ euros par exemplaire et par jour de retard. La bibliothèque souhaite mettre en place un système informatique sommaire pour gérer les prêts à ses adhérents et les pénalités dues. Pour ce faire, on attribue à chaque adhérent un numéro d'adhérent auquel on associe son nom, son adresse, ainsi que la liste des exemplaires empruntés, avec pour chaque exemplaire, sa date limite de retour théorique (date d'emprunt + 21 jours). On utilise les déclarations suivantes :

```
1  enum class Categorie {livre, BD, journal};
2
3  struct Exemplaire {
4    int nref; // le numéro de référence
5    Categorie cat;
6    string titre;
7    Date dretour; // date limite de retour
8  };
9
10 struct Adherent {
11   int nA; // le numéro de l'adherent
12   string nom;
13   string adresse;
14   vector<Exemplaire> pret;
15 };
```

On suppose de plus déjà défini et écrit par ailleurs, un type structuré `Date` avec les opérateurs suivants (où `d, d1` sont des `Date` et `i` est un entier) :

- `d + i`, ajoute un entier à une date et retourne la date avancée de cet entier ;
- l'opérateur `d1 - d`, qui calcule la différence entre `d1` et la date de retour `d`. Il retourne 0 si `d1` précède ou est égale à `d` (la date de retour `d` n'a pas été dépassée), et le nbre de jours de dépassement `d1 - d` sinon
- L'operateur d'affichage d'une date.

On ne demande pas d'écrire les fonctions de manipulation des dates.

Le but de ce problème est de gérer les emprunts et d'écrire des lettres de rappels en cas de retard trop grand. Voici un exemple succinct de lettre de rappel (on ne demande pas de faire une belle présentation) :

```
Lettre de rappel de la bibliothèque
No : 42, Nom : Florent, Addr : 2, rue de nulle part
  Ref : 2324, Cat : livre, Titre : L'outsider, Retour : 12/03/2022
  Ref : 5351, Cat : BD, Titre : Isnogood, Retour : 02/05/2022
  Ref : 1242, Cat : journal, Titre : Pour la Science, Retour : 14/04/2022
Suite à vos retard, vous avez une pénalite de 45 Euros
```

On vous demande d'écrire les fonctions suivantes :

1. L'opérateur d'affichage qui imprime une catégorie.

```
162 ostream& operator<<(ostream& out, Categorie c) {
163   switch (c) {
164     case Categorie::livre : out << "livre"; break;
165     case Categorie::BD : out << "BD"; break;
166     case Categorie::journal : out << "journal"; break;
167   }
168   return out;
169 }
```

2. L'opérateur d'affichage qui imprime les informations relatives à un exemplaire donné.

```
172 ostream& operator<<(ostream& out, Exemplaire e) {
173   out << "Ref : " << e.nref << ", Cat : " << e.cat
174       << ", Titre : " << e.titre << ", Retour : " << e.dretour;
175   return out;
176 }
```

3. L'opérateur d'affichage qui, étant donné un adhérent, imprime son nom et son adresse, ainsi que les informations relatives aux exemplaires empruntés.

```
179 ostream& operator<<(ostream& out, Adherent a) {
180   out << "No : " << a.nA << ", Nom : " << a.nom
181       << ", Addr : " << a.adresse << endl;
182   for (int i = 0; i < a.pret.size(); i++) out << "  " << a.pret[i] << endl;
183   return out;
184 }
```

4. La procédure `void emprunter(Adherent &a, Exemplaire e, Date d)` qui étant donnés un adhérent, une date du jour, et un exemplaire permet d'ajouter l'exemplaire donné à la liste des emprunts de cet adhérent, s'il n'a pas déjà atteint son nombre d'emprunts maximum. Dans le cas contraire, on signalera une exception de type `runtime_error` avec le message «Emprunt interdit».

```
188 void emprunter(Adherent &a, Exemplaire e, Date d) {
189   if (a.pret.size() >= 10)
190     throw runtime_error("Emprunt interdit");
191   e.dretour = d + 21;
192   a.pret.push_back(e);
193 }
```

5. La fonction `calculPenalite` qui étant donnés un adhérent et une date du jour, retourne la somme due par cet adhérent (éventuellement 0).

```
197 float calculPenalite(Adherent a, Date d) {
198   int njour = 0;
199   for (int i = 0; i < a.pret.size(); i++)
200     njour += d - a.pret[i].dretour;
201   return 0.20*njour;
202 }
```

6. Proposez un type permettant de représenter l'ensemble des adhérents de la bibliothèque.

```
206 using Biblio = vector<Adherent>;
```

7. La procédure `lettresDeRappel` utilisant les fonctions précédentes et qui permet, pour une bibliothèque et à une date donnée, d'afficher à l'écran les lettres de rappel à envoyer à chaque adhérent de la bibliothèque ayant plus de 15 euros de pénalité. Pour chaque retard, la fonction affiche l'adhérent (avec les exemplaires qu'il détient encore) ainsi que le montant de sa pénalité.

```
210 void lettresDeRappel(Biblio b, Date d) {
211   for (Adherent a : b) { // Ou mieux, for (const Adherent &a : b)
212     float p = calculPenalite(a, d);
213     if (p > 15) {
214       cout << "Lettre de rappel de la bibliothèque " << endl;
215       cout << a << "Suite à vos retard, vous avez une pénalite de "
216            << p << " Euros " << endl;
217    }
218   }
219 }
```

8. La procédure `retour` qui étant donnés un adhérent, une date du jour, et la référence d'un exemplaire permet de retirer l'exemplaire considéré de la liste des emprunts de cet adhérent. La fonction devra retourner l'éventuel montant des pénalités de retard correspondant à cet exemplaire. Si la référence ne correspond à aucun exemplaire emprunté, on signalera une exception.

```
223 int cherche(Adherent &a, int ref) {
224   for (int i = 0; i < a.pret.size(); i++)
225     if (a.pret[i].nref == ref) return i;
226   return -1;
227 }
228 float retour(Adherent &a, Date d, int ref) {
229   int pos = cherche(a, ref);
230   if (pos == -1) throw runtime_error("Reference non trouvée");
231   Exemplaire e = a.pret[pos];
232   float p = (d - a.pret[pos].dretour)*0.2;
233   a.pret[pos] = a.pret.back(); // on remplace par le dernier emprunté.
234   a.pret.pop_back();
235   return p;
236 }
```