ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE – LAUSANNE
POLITECNICO FEDERALE – LOSANNA
SWISS FEDERAL INSTITUTE OF TECHNOLOGY – LAUSANNE

Faculté Informatique et Communications
Cours de Programmation Orientée Objet, sections MA et PH
Chappelier J.-C.

Programmation Orientée Objet (SMA/SPH) : Correction examen final

1er juin 2023

INSTRUCTIONS (à lire attentivement)

IMPORTANT ! Veuillez suivre les instructions suivantes à la lettre sous peine de voir votre examen annulé dans le cas contraire.

1. Vous disposez d'une heure quarante-cinq minutes pour faire cet examen (9h15 – 11h00).
2. Vous devez **écrire à l'encre noire ou bleu foncée**, pas de crayon ni d'autre couleur.
   N'utilisez **pas non plus de stylo effaçable** (perte de l'information à la chaleur).
3. Vous avez droit à toute documentation papier.
   En revanche, vous ne pouvez pas utiliser d'ordinateur personnel, ni de téléphone portable, ni aucun autre matériel électronique.
4. Répondez aux questions directement sur la donnée ; ne joignez aucune feuille supplémentaire ; **seul ce document sera corrigé**.
   Vous disposez, si nécessaire, d'une page blanche supplémentaire en fin de sujet.
5. Lisez attentivement et *complètement* les questions de façon à ne faire que ce qui vous est demandé. Si l'énoncé ne vous paraît pas clair, ou si vous avez un doute, demandez des précisions à l'un(e) des assistant(e)s.
6. L'examen comporte quatre exercices indépendants, qui peuvent être traités dans n'importe quel ordre, mais qui ne rapportent pas la même chose (les points sont indiqués, le total est de 132) :
   1. questions courtes : 27 points ;
   2. conception : 44 points ;
   3. correction d'erreurs : 29 points ; et
   4. déroulement de programme : 32 points.

   Tous les exercices comptent pour la note finale.

Question 1 – Questions diverses [27 points]

1.1 Quand on arrive en ville [4 points]

Considérez le code ci-dessous et cochez clairement la ou les proposition(s) correcte(s) ; pénalités pour réponses fausses sélectionnées.

```
#include <string>
using namespace std;

class Ville {
  string nom;
public:
  Ville(const string nom = "Lausanne");
  string getNom() const { return nom; }
};

int main() {
  return 0;
}
```

[ ] Ce code ne compile pas.

[ ] `Ville` est une classe abstraite.

[ ] `Ville` n'a qu'un seul constructeur.

[ ] Ce code instancie `Ville`.

[ ] Toutes les `Ville` ont pour nom `"Lausanne"`.

[**X**] Il est impossible de changer le nom d'une instance de `Ville`.

Justifiez *brièvement* chaque réponse (cochée ou non) :

- Ce code compile même si pour le moment il ne fait rien.
- Il n'y a aucune méthode virtuelle (donc encore moins virtuelle pure).
- Il y a (au moins) deux constructeurs : un par défaut (avec `"Lausanne"`, donc) et un avec argument `string`. Le fait qu'il(s) ne soi(en)t pas défini(s) ici n'est pas un problème (cf point suivant).
  Et il a aussi (au moins) le contructeur de copie fourni par défaut.
- Aucune instance n'est crée (le `main()` ne fait strictement rien).
- cf constructeurs
- Il n'y a pas de « *setter* » et `nom` est en `private`.

1.2 H ? Tag ? ? [3 points]

À quoi sert un fichier `.h` ?

Réponse : Cela sert à fournir ce qui est nécessaire (types, prototypes) aux **autres** `.cc` pour compiler.

1.3 T'as pas oublié les piles ? [10 points]

Considérez la classe ci-dessous conçue pour implémenter une pile (on considère que tous les `#include` nécessaires ont été faits) :

```
class Stack {
public:
  // ...votre code ira ici...
private:
  size_t size;
  std::array<const Data*, 32> content;
};
```

L'attribut `size` indique le nombre d'éléments effectivement empilés (et donc aussi, *indirectement*, l'index du dernier élément empilé).

La `Stack` n'est pas responsable de son contenu, elle ne sert qu'à référencer des éléments extérieurs existants par ailleurs.

[2 points] Définissez un constructeur par défaut pour `Stack`, qui crée une pile vide :

```
Stack() : size(0) {}
```

Pas besoin d'initialiser le tableau (puisqu'avec un attribut `size` à `0` on n'utilisera pas son contenu), mais ce n'est pas faux de le faire.

[2 points] Définissez une méthode `empty()` permettant de tester si une pile est vide :

```
bool empty() const { return size == 0; }
```

Ne pas oublier le `const` !

[3 points] Définissez une méthode `push()` qui reçoit une référence constante sur une `Data` et l'empile. Cette méthode retourne un `bool` indiquant si l'insertion a pu être réalisée ou non.

```
bool push(const Data& d) {
  if (size >= content.size()) return false;
  content[size++] = &d;
  return true;
}
```

Jamais de « constante en dur » dans le code : `content.size()` et non pas `32` !

[3 points] Définissez enfin une méthode `pop()` qui extrait le dernier élément de la pile et le retourne par référence constante. Cette fonction lance une exception (de votre choix) si la pile est vide.

```
const Data& pop() {
  if (empty()) throw "pop on empty Stack";
  return *content[--size];
}
```

suite au dos ☞

1.4 Top Gear [6 points]

Considérez le code des classes ci-dessous, contenu dans le fichier `voiture.h`. Ce code est correct.

```
class Voiture {
public:
  Voiture(double c) : cylindree(c) {}
  virtual ~Voiture() = default;
  virtual double couple() { return cylindree * NmParLitre; }
private:
  static constexpr double NmParLitre = 0.95;
  double cylindree;
};

class VoitureTurbo : public Voiture {
public:
  using Voiture::Voiture;
  double couple() { return Voiture::couple() * Boost; }
private:
  static constexpr double Boost = 1.75;
};
```

Pour chacun des fichiers donnés à gauche ci-dessous, cochez clairement la ou les proposition(s) correcte(s) proposée(s) à sa droite ; pénalités pour réponses fausses sélectionnées.

Justifiez *très brièvement* votre réponse à chaque fois.

Note : $0.95 \times 1.5 = 1.425$ et $1.75 \times 1.425 = 2.49375$.

```
#include <iostream>
#include "voiture.h"
using namespace std;
int main() {
  Voiture v(1.5);
  cout << "Couple de " << v.couple()
    << "Nm, cylindrée de "
    << v.cylindree << "L." << endl;
  return 0;
}
```

- [ ] `Voiture` est une classe abstraite.
- [X] Le code ne compile pas.
- [ ] Le code compile, mais produit une erreur si on l'exécute.
- [ ] Le code compile et s'exécute correctement ; il affiche : `Couple de 1.425Nm, cylindrée de 1.5L.`
- [ ] Le code compile et s'exécute correctement ; il affiche `Couple de 2.49375Nm, cylindrée de 1.5L.`

Brève justification : `cylindree` est `private`.

```
#include <iostream>
#include "voiture.h"
using namespace std;
int main() {
  VoitureTurbo v1(1.5);
  Voiture& v2(v1);
  cout << v2.couple() << endl;
  return 0;
}
```

- [ ] `VoitureTurbo` est une classe abstraite.
- [ ] Le code ne compile pas.
- [ ] Le code compile, mais produit une erreur si on l'exécute.
- [ ] Le code compile et s'exécute correctement ; il affiche `1.425`
- [X] Le code compile et s'exécute correctement ; il affiche `2.49375`

Brève justification : `virtual` et référence : donc polymorphisme.

```
#include <iostream>
#include "voiture.h"
using namespace std;
int main() {
  const VoitureTurbo v1(1.5);
  const Voiture& v2(v1);
  cout << v2.couple() << endl;
  return 0;
}
```

[X] Le code ne compile pas.

[ ] Le code compile, mais produit une erreur si on l'exécute.

[ ] Le code compile et s'exécute correctement ; il affiche `1.425`

[ ] Le code compile et s'exécute correctement ; il affiche `2.49375`

Brève justification : `couple()` n'est pas `const`.

```
#include <iostream>
#include "voiture.h"
using namespace std;
int main() {
  VoitureTurbo* ptr(new Voiture(1.5));
  cout << ptr->couple() << endl;
  delete ptr;
  return 0;
}
```

[X] Le code ne compile pas.

[ ] Le code compile, mais produit une erreur si on l'exécute.

[ ] Le code compile et s'exécute correctement ; il affiche `1.425`

[ ] Le code compile et s'exécute correctement ; il affiche `2.49375`

Brève justification : Une `VoitureTurbo` est une `Voiture`, **mais** toutes les `Voiture` ne sont pas nécessairement des `VoitureTurbo` : on ne peut pas faire l'affectation dans ce sens.

1.5 Keskidi ? [4 points]

Lors de la création d'un projet de programmation, vous obtenez l'erreur suivante :

```
/usr/bin/ld : polonaise.o : dans la fonction « eval » :
polonaise.cc:33 : référence indéfinie vers « Vecteur::Vecteur »
/usr/bin/ld : polonaise.cc:39 : référence indéfinie vers « Vecteur::add »
collect2: error: ld returned 1 exit status
```

① Est-ce une erreur de compilation ou d'édition de liens ?

② Que signifie-t-elle (clairement, en français) ?

③ Comment la corriger ? (quel(s) fichier(s) modifier et comment ?)

① C'est une erreur d'édition de liens...

② ...qui dit que le code (*objet*) de certains morceaux (le constructeur de `Vecteur` et sa méthode `add()`) sont manquants ;
il manque donc visiblement un fichier `.o` dans cette édition de liens (typiquement `vecteur.o`, vus les noms).

③ Il faut corriger la commande d'édition de liens pour ajouter les parties manquantes ; très certainement : dans le `Makefile`, ajouter `vecteur.o` à la liste des dépendances de la cible (exécutable) qui était en train d'être créer.
Cela n'a en tout cas rien à voir avec l'inclusion de fichiers d'entête (lesquels sont utiles à la *compilation*, pas à l'édition de liens !).

suite au dos ☞

Question 2 – Conception [44 points]

On souhaite écrire un programme pour gérer des véhicules. Les deux concepts de base sont des personnes (type `Personne`) et des véhicules (type `Vehicule`). Ces deux types possèdent chacun un nom, qui est fixé une fois pour toute au départ, que l'on peut consulter et que l'on peut afficher (via l'opérateur usuel).

Par ailleurs :

- les personnes possèdent un ensemble de véhicules, possiblement vide ; elles ne peuvent par contre pas posséder deux fois le *même* véhicule (mais des véhicules de même nom, oui, pas de problème) ;
- les véhicules ont (au moins) un chauffeur et un propriétaire ; ceux-ci peuvent changer et ne sont pas connus au départ.

Les seuls véhicules possibles sont des motos, lesquelles ont en plus un (et un seul) passager, et des voitures, lesquelles ont entre exactement 1 et 7 passagers. Le nombre de passagers d'une voiture doit être donné au départ (3 par défaut [1]) et ne pourra plus changer.

Enfin, on souhaite que les personnes puissent acheter, vendre et conduire des véhicules.

En guise d'illustration, on pourrait par exemple avoir le `main()` suivant :

```
int main()
{
  Personne Pierre("Pierre");
  Personne Jeanne("Jeanne");

  Voiture L4("4L"); // 4 places
  Voiture CV2("2CV", 5); // 5 places

  Pierre.achete(L4);
  Jeanne.achete(CV2);
  Jeanne.conduit(L4);

  cout << Pierre << endl;
  cout << Jeanne << endl;

  return 0;
}
```

qui pourrait typiquement afficher quelque chose comme :

```
"Pierre" possède les véhicules suivants :
    - une voiture 4 places "4L", proprio : Pierre, chauffeur : Jeanne
      sans passager

"Jeanne" possède les véhicules suivants :
    - une voiture 5 places "2CV", proprio : Jeanne, sans chauffeur,
      sans passager
```

1. Donc 4 places en tout, avec le chaufeur.

① **[30.5 points] Conception**

Sans donner tout le détail du code complet (on ne demande ici qu'une *conception*), écrivez **en C++** les classes, les éventuelles relations d'héritage, les attributs et les méthodes des classes, les droits d'accès et les éventuelles fonctions (externes) que vous utiliseriez pour implémenter un tel programme.

Précisez les types des attributs et les prototypes des méthodes/fonctions, mais ne donnez pas leur définition (on répète : il s'agit ici de la partie *conception*, pas de l'implémentation ; c.-à-d. les prototypes, pas les définitions des méthodes).

Lorsque c'est nécessaire, indiquez également les constructeurs et les destructeurs (sans leur définition).

② **[6 points] Programmation 1**

Implémentez (c.-à-d. définissez) la méthode permettant à une personne d'acheter un véhicule. Si cette méthode fait appel à d'autres méthodes (de la même classe ou d'autres classes), définissez aussi toutes ces autres méthodes.

Acheter un véhicule ce n'est pas le voler : il faut que le véhicule soit disponible : soit sans propriétaire, soit que le propriétaire accepte de le vendre.
Pour simplifier, on supposera que chaque personne accepte de vendre un véhicule dont elle est effectivement le propriétaire.

③ **[7.5 points] Programmation 2**

Implémentez (c.-à-d. définissez) tout ce qui permet d'afficher une voiture (voir l'exemple donné page ci-contre).

---

① Ci-contre *une* réponse possible pour la conception.
En particulier toutes les méthodes-outils données ici ne sont pas forcément strictement nécessaires. Cela dépend de la conception et de la façon dont le code pour les questions ② et ③ a été écrit.

J'ai mis tous les attributs en `private`, mais on peut les tolérer en `protected`.

Pour toutes les classes : s'il y a des attributs, il faut le constructeur pour les initialiser ; s'il n'y a pas d'attribut, le contructeur par défaut par défaut suffit, **sauf** pour les sous-classes dont la super-classe a un constructeur spécifique.

```
class ObjetNomme {
public:
  ObjetNomme(const string& nom);
  const string& nom() const;
  virtual void affiche(ostream& out) const;
private:
  const string nom_;
};

ostream& operator<<(ostream& out, const ObjetNomme& o);

class Vehicule;

class Personne : public ObjetNomme {
public:
  Personne(const string& nom);
  bool achete(Vehicule& v);
  bool vend(Vehicule& v);
  void conduit(Vehicule& v); // can be const
  virtual void affiche(ostream& out) const override;
private:
  set<Vehicule*> garage;
};

class Vehicule : public ObjetNomme {
public:
  Vehicule(const string& nom);
  void est_vendue_par(const Personne& p);
  bool est_achete_par(Personne& p);
  void est_conducteur(const Personne& p);
  virtual void affiche_type(ostream& out) const = 0;
  virtual void affiche(ostream& out) const override;
private:
  Personne* proprietaire;
  const Personne* chauffeur;
};

class Moto : public Vehicule {
public:
  Moto(const string& nom);
  virtual void affiche_type(ostream& out) const = 0;
  virtual void affiche(ostream& out) const override;
private:
  Personne* passager;
};

class Voiture : public Vehicule {
public:
  Voiture(const string& nom, size_t nb_places = 4);
  virtual void affiche_type(ostream& out) const = 0;
  virtual void affiche(ostream& out) const override;
private:
  const size_t nb_passagers;
  array<Personne*, 7> passagers;
};
```

② Voici un exemple possible, sans fuite d'encapsulation et avec maintient de la cohérence des données :

```
bool Personne::achete(Vehicule& v)
{
  if (v.est_achete_par(*this)) {
    garage.insert(&v);
    return true; // they don't know about set::insert() return value
  }
  return false;
}

bool Vehicule::est_achete_par(Personne& p)
{
  // shall check whether vehicule is available for sale or not
  if ((proprietaire == nullptr) or
      proprietaire->vend(*this)) {
    proprietaire = &p;
    return true;
  }
  return false;
}

bool Personne::vend(Vehicule& v)
{
  if (garage.contains(&v)) {
    garage.erase(&v);
    v.est_vendue_par(*this);
    return true;
  }
  return false;
}

void Vehicule::est_vendue_par(const Personne& p)
{
  if (proprietaire == &p) proprietaire = nullptr;
}
```

③ Voici un exemple possible, sans fuite d'encapsulation et avec « généricité » (polymorphisme) de l'affichage :

```
void ObjetNomme::affiche(ostream& out) const
{ out << '"' << nom_ << '"'; }

ostream& operator<<(ostream& out, const ObjetNomme& o)
{
  o.affiche(out);
  return out;
}

void Vehicule::affiche(ostream& out) const
{
  affiche_type(out);
  ObjetNomme::affiche(out);
  if (proprietaire != nullptr)
    out << ", proprio : " << proprietaire->nom();
  if (chauffeur != nullptr)
    out << ", conduit par : " << chauffeur->nom();
  out << endl;
}

void Voiture::affiche_type(ostream& out) const
{ out << "une voiture " << nb_passagers+1 << " places "; }

void Voiture::affiche(ostream& out) const
{
  Vehicule::affiche(out);
  out << "\tpassagers :" << endl;
  for (auto p : passagers)
    if (p != nullptr) out << "\t- " << p->nom() << endl;
}
```

suite au dos ☞

Question 3 – Houston, on a un problème [29 points]

Voici le code corrigé : en bleu les erreurs de compilation en vert celle de l'édition de liens et en rouge celle de l'exécution.

```
1   #include <iostream>
2   #include <vector>
3   #include <memory>
4   using namespace std;
5
6   enum Meteo { Degage, Orage };
7
8   class Astronaute {
9     const string nom;
10  public:                                  ⑧ = 0
11    Astronaute(string nom_): nom{nom_} {}
12    virtual string nationalite() const;
13    const string& getNom() { return nom; }
14  };                        ③ const
15
16  class Cosmonaute: public Astronaute {
17    using Astronaute::Astronaute;
18    string nationalite() override {
19      return "Russe"; } ① const
20  };
21
22  class Vol {
23  public:
24    Astronaute const &a1, &a2;
25    Vol(const Astronaute &a1_,
26        const Astronaute &a2_)
27        { a1=a1_; a2=a2_; }  ② section deux-points
28    virtual void decoller(Meteo) = 0;
29  };          ④ bool
30
31  ostream& operator<<(ostream& o, Vol& v) {
32    o << "Vol[a1=" << v.a1.getNom()
33      << ",a2=" << v.a2.getNom() << "]";
34  }
35 ⑦ return o;
36    class VolTest: public Vol {
37      using Vol::Vol;
38      bool decoller(Meteo) override {
39        return true;
40      }
41    };
```

```
42
43  class VolReel: public Vol {
44    using Vol::Vol;
45    bool decoller(Meteo m) override {
46      return m == Degage;
47    }
48  };
49
50  class ProgrammeSpatial {
51    vector<unique_ptr<Vol>> vols;
52  public:
53    void ajouter_vol(Vol* vol) {
54     vols.push_back(unique_ptr<Vol>(vol));
55    }
56
57    bool demarrer(Meteo m) {
58      for (auto& v : vols) {
59        if (!v.decoller(m)) {
60          return false;
61        }     ⑤ ->
62      }
63      return true;
64    }
65  };
66                                  ⑥ ne rien mettre
67  int main() {
68    ProgrammeSpatial programme();
69    Cosmonaute a1("Youri Gagarine");
70    Cosmonaute a2("Valentina Terechkova");
71    VolTest v1(a1, a2);
72    programme.ajouter_vol(&v1);
73    if (programme.demarrer(Degage))
74      return 0;
75    return 1;
76  }
```

```
VolTest* pv1 = new VolTest(a1, a2);
programme.ajouter_vol(pv1);
```

Pour les erreurs de compilation (dans l'ordre détectées par le compilateur) :

① **l.18 :** il faut rendre la méthode `nationalite` compatible avec celle de `Astronaute` ; vu le contexte, il est évident qu'elle doit être `const` ;

② **l.28 :** on ne peut pas affecter des références ; le seul moyen est de les initialiser (donc utiliser la « section deux-points ») ;

③ **l.13 :** il est impératif que cette méthode soit `const` ;

④ **l.29 :** il faut rendre la méthode `decoller` compatible avec celles de ses sous-classes ; vu le contexte, il est évident qu'elle doit retourner un `bool` ;

⑤ **l.59 :** `v` est un pointeur, donc `->` au lieu de `.` (ou alors `(*v).`, avec les parenthèses) ;

⑥ **l.68 :** ceci n'est pas une instance d'un `ProgrammeSpatial`, mais une *fonction* qui retourne un `ProgrammeSpatial` ;

⑦ **l.35 :** celle-ci est peut être la plus difficile à trouver car le compileur *peut* la détecter si l'on compile avec *warning* (ce qui n'a pas été le cas) : il est impératif d'avoir un `return` pour une fonction non-`void` (ou alors la rendre `void`, mais ici c'est contraire au standard et empêcherait l'enchaînement des affichages).

⑧ L'erreur d'édition de liens vient du fait que la méthode virtuelle `Astronaute::nationalite()` n'est pas définie (**l.12**). Il suffit de le faire ou alors de la rendre virtuelle pure (en ajoutant `= 0`) (les *deux* solution sont valides, au choix).

⑨ L'erreur d'exécution vient du fait que l'on passe une variable allouée *statiquement*, alors que le modèle suppose que le `ProgrammeSpatial` soit « propriétaire » de ses vols (puisqu'il a des `unique_ptr`). Il est donc impératif ici de faire de l'allocation dynamique (ou de changer complètement le modèle ; plus compliqué).

suite au dos ☞

Question 4 – Exécution de programme [32 points]

Le programme ci-contre compile et s'exécute sans erreurs. Qu'affiche-t-il ?
Répondez-ci dessous puis justifiez votre réponse en expliquant (p.ex. à droite de votre affichage) les points *importants* (on ne vous demande pas ici de paraphraser le code, mais bien de montrer que vous avez compris ce qui se passe !).

Réponses :

```
#1 ===========
Création de ???
New LS1 de James S.
Signé
Un Signé
Livre
Un Signé
Dtr. LS de James S.
Dtr. de ???
#2 ===========
Création de Si une nuit...
New LS2 de Italo C.
Création de Loin de Malbork
New LS2 de Tazio B.
Création de Si une nuit...
Création de Si une nuit...
New LS2 de Italo C.
Création de Loin de Malbork
New LS2 de Tazio B.
Si une nuit... contient Loin de Malbork
Imbrq.
Un Imbrq.
-----
Livre
Un Imbrq.
-----
Livre
Un Imbrq.
Dest. de livres
Dtr. LS de Tazio B.
Dtr. de Loin de Malbork
Dtr. LS de Italo C.
Dtr. de Si une nuit...
Dtr. de Si une nuit...
Dtr. LS de Tazio B.
Dtr. de Loin de Malbork
Dtr. LS de Italo C.
Dtr. de Si une nuit...
```

Réponses, ligne à ligne (de l'affichage) :
étape 1 :
constructeur `Livre` par défaut (l.84 → l.31 → l.8)
constructeur `LivreSigne` (l.84 → l.32)
`LivreSigne::type` (l.85 → l.43)
`LivreSigne::affiche` (l.86 → l.46 → l.43)
`Livre::type()` (l.88 → l.14)
`LivreSigne::affiche` (l.89 → **l.46** (**polymorphisme**) → l.43)
destruction de `l1` : destructeur `LivreSigne` (l.89–fin → l.39 → l.40)
destruction de `l1` : destructeur `Livre` : 1 (l.89–fin → l.39 → l.11+12)

étape 2 :
construction de `l1` :
- 1er constructeur `Livre`
- 1er constructeur `LivreSigne`

construction de `l2` :
- 2e constructeur `Livre`
- 2e constructeur `LivreSigne`

construction de `l3` :
- 3e constructeur `Livre`
- construction de la copie de `l1` :
  - 4e constructeur `Livre`
  - 3e constructeur `LivreSigne`
- construction de la copie de `l2` :
  - 5e constructeur `Livre`
  - 4e constructeur `LivreSigne`
- affichage du message du corps (du constructeur de `LivreImbrique`)

`LivreImbrique::type`
`LivreImbrique::affiche`
`Livre::type`
`LivreImbrique::affiche` (**polymorphisme**)
`Livre::type`
`LivreImbrique::affiche` (**polymorphisme**)
destructeur `LivreImbrique`
destructeur `LivreSigne` 4
destructeur `Livre` 5
destructeur `LivreSigne` 3
destructeur `Livre` 4
destructeur `Livre` 3
destructeur `LivreSigne` 2
destructeur `Livre` 2
destructeur `LivreSigne` 1
destructeur `Livre` 1

```cpp
#include <iostream>
#include <vector>
#include <memory>
using namespace std;

class Livre {
public:
  Livre(string const& t = "???"): titr_(t)
  { cout << "Création de " << t << endl; }

  virtual ~Livre()
  { cout << "Dtr. de " << titr_ << endl; }

  string type() const { return "Livre"; }

  virtual void affiche() const
  { cout << "Un " << type() << endl; }

  virtual Livre* copie() const
  { return new Livre(titr_); }

  const string& titre() const
  { return titr_; }

private:
  const string titr_;
};

class LivreSigne : public Livre {
public:
  LivreSigne(string const& a): auteur(a)
  { cout << "New LS1 de " << a << endl; }

  LivreSigne(string const& a,
             string const& t)
  : Livre(t), auteur(a)
  { cout << "New LS2 de " << a << endl; }

  ~LivreSigne()
  { cout << "Dtr. LS de " << auteur
         << endl;                        }

  string type() const { return "Signé"; }

  void affiche() const
  { cout << "Un " << type() << endl; }
private:
  string const auteur;

  virtual Livre* copie() const
  { return new LivreSigne(auteur,
                          titre());  }
};

class LivreImbrique : public Livre {
public:
  LivreImbrique(Livre const& l1,
                Livre const& l2)
  : Livre(l1.titre()),
    contenant(l1.copie()),
    contenu(l2.copie())
  { cout << l1.titre() << " contient "
         << l2.titre() << endl;          }

  ~LivreImbrique()
  { cout << "Dest. de livres" << endl; }

  string type() const { return "Imbrq."; }

  void affiche() const
  { cout << "Un " << type() << endl; }

private:
  unique_ptr<Livre> contenant;
  unique_ptr<Livre> contenu;

  virtual Livre* copie() const
  { return new LivreImbrique(*contenant,
                             *contenu);  }
};

int main() {
  { cout << "#1 ===========" << endl;
    LivreSigne l1("James S."); // Seward
    cout << l1.type() << endl;
    l1.affiche();
    Livre& l2(l1);
    cout << l2.type() << endl;
    l2.affiche();                        }
  { cout << "#2 ===========" << endl;
    LivreSigne l1("Italo C.", // Calvino
      "Si une nuit...");
    LivreSigne l2("Tazio B.", // Bazakbal
      "Loin de Malbork");
    const LivreImbrique l3(l1, l2);
    cout << l3.type() << endl;
    l3.affiche(); cout << "-----" << endl;
    const Livre& l4(l3);
    cout << l4.type() << endl;
    l4.affiche(); cout << "-----" << endl;
    const Livre* l5(&l3);
    cout << l5->type() << endl;
    l5->affiche();                       }
  return 0;
}
```