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.

NOM : Hanon Ymous
(000000)
Place : 0

#0000

Programmation Orientée Objet (SMA/SPH) : 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"`.
- [ ] Il est impossible de changer le nom d'une instance de `Ville`.

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

1.2 H ? Tag ? ? [3 points]

À quoi sert un fichier `.h` ?

Réponse :

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 :

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

[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.

[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.

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;
}
```

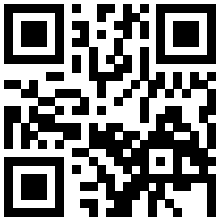
- [ ] `Voiture` est une classe abstraite.
- [ ] Le code ne compile pas.
- [ ] Le code compile, mais produit une erreur si on l'exécute.
- [ ] Le code compile et s'exécute correctement ; il affiche : `Couple de 1.425Nm, cylindrée de 1.5L.`
- [ ] Le code compile et s'exécute correctement ; il affiche `Couple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

Brève justification :

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

Brève justification :



```
#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;
}
```

[ ] 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 :

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

[ ] 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 :

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 ?)

Réponses :

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).

Suite des réponses à la Question 2 :

Suite des réponses à la Question 2 :

Ne pas écrire dans cette zone.

suite au dos ☞

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

Le programme fourni ci-dessous contient neuf erreurs, dont sept sont détectées par le compilateur (p. 11), une lors de l'édition de liens (p. 12), et une a lieu lors de l'exécution du programme (p. 12).

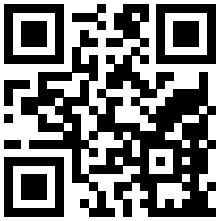
On vous demande de corriger ces erreurs en les marquant directement sur le code ci-dessous, ou juste en dessous.

Attention ! On ôtera 2 points pour toute indication d'une erreur qui n'en est pas une.

Afin de faciliter votre réponse/lecture, nous fournissons à nouveau le même code en page 13. Vous pouvez aussi répondre là-bas, ainsi que donner des explications complémentaires en bas de pages 12 et 13, ou même directement à coté des messages d'erreur.

```
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:
11    Astronaute(string nom_): nom{nom_} {}
12    virtual string nationalite() const;
13    const string& getNom() { return nom; }
14  };
15
16  class Cosmonaute: public Astronaute {
17    using Astronaute::Astronaute;
18    string nationalite() override {
19      return "Russe"; }
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_; }
28    virtual void decoller(Meteo) = 0;
29  };
30
31  ostream& operator<<(ostream& o, Vol& v) {
32    o << "Vol[a1=" << v.a1.getNom()
33      << ",a2=" << v.a2.getNom() << "]";
34  }
35
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        }
62      }
63      return true;
64    }
65  };
66
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  }
```

Ne pas écrire dans cette zone.



Messages du compilateur :

```
astro.cc:18:24: error: non-virtual member function marked 'override' hides virtual member function
  string nationalite() override {
                       ^
astro.cc:12:18: note: hidden overloaded virtual function 'Astronaute::nationalite' declared here: different
qualifiers ('const' vs unqualified)
  virtual string nationalite() const;
                 ^
astro.cc:25:3: error: constructor for 'Vol' must explicitly initialize the reference member 'a1'
  Vol(const Astronaute &a1_,
  ^
astro.cc:24:21: note: declared here
  Astronaute const &a1;
                    ^
astro.cc:26:3: error: constructor for 'Vol' must explicitly initialize the reference member 'a2'
  Vol(const Astronaute &a1_,
  ^
astro.cc:25:21: note: declared here
  Astronaute const &a2;
                    ^
astro.cc:28:10: error: no viable overloaded '='
      { a1=a1_; a2=a2_; }
        ~~^~~~
astro.cc:8:7: note: candidate function (the implicit copy assignment operator) not viable: 'this' argument
has type 'const Astronaute', but method is not marked const
class Astronaute {
      ^
astro.cc:28:18: error: no viable overloaded '='
      { a1=a1_; a2=a2_; }
                ~~^~~~
astro.cc:8:7: note: candidate function (the implicit copy assignment operator) not viable: 'this' argument
has type 'const Astronaute', but method is not marked const
class Astronaute {
      ^
astro.cc:33:21: error: 'this' argument to member function 'getNom' has type 'const Astronaute', but function
is not marked const
  o << "Vol[a1=" << v.a1.getNom()
                     ^~~~
astro.cc:13:17: note: 'getNom' declared here
  const string& getNom() { return nom; }
                ^
astro.cc:34:18: error: 'this' argument to member function 'getNom' has type 'const Astronaute', but function
is not marked const
    << ",a2=" << v.a2.getNom() << "]";
                 ^~~~
astro.cc:13:17: note: 'getNom' declared here
  const string& getNom() { return nom; }
                ^
astro.cc:39:8: error: virtual function 'decoller' has a different return type ('bool') than the function it
overrides (which has return type 'void')
  bool decoller(Meteo) override {
  ~~~~ ^
astro.cc:28:16: note: overridden virtual function is here
  virtual void decoller(Meteo) = 0;
          ~~~~ ^
astro.cc:46:8: error: virtual function 'decoller' has a different return type ('bool') than the function it
overrides (which has return type 'void')
  bool decoller(Meteo m) override {
  ~~~~ ^
```

```
astro.cc:29:16: note: overridden virtual function is here
  virtual bool decoller(Meteo) = 0;
         ~~~~ ^
astro.cc:60:13: error: no member named 'decoller' in 'std::unique_ptr<Vol>'; did you mean to use '->' instead
of '.'?
      if (!v.decoller(m)) {
            ^
            ->
astro.cc:60:11: error: invalid argument type 'void' to unary expression
      if (!v.decoller(m)) {
          ^~~~~~~~~~~~~~
astro.cc:69:29: warning: empty parentheses interpreted as a function declaration [-Wvexing-parse]
  ProgrammeSpatial programme();
                           ^~
astro.cc:69:29: note: replace parentheses with an initializer to declare a variable
  ProgrammeSpatial programme();
                           ^~
                           {}
astro.cc:73:3: error: base of member reference is a function; perhaps you meant to call it with no arguments?
  programme.ajouter_vol(&v1);
  ^~~~~~~~~
          ()
astro.cc:74:7: error: base of member reference is a function; perhaps you meant to call it with no arguments?
  if (programme.demarrer(Degage))
      ^~~~~~~~~
               ()
```

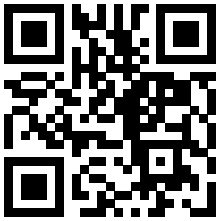
Messages de l'éditeur de liens (une fois les sept erreurs de compilation corrigées) :

```
/usr/bin/ld: /tmp/ccMLpjbY.o: in function `Astronaute::Astronaute(std::__cxx11::basic_string<char,
 std::char_traits<char>, std::allocator<char> >)':
astro.cc:11: undefined reference to `vtable for Astronaute'
/usr/bin/ld: /tmp/ccMLpjbY.o: in function `Astronaute::~Astronaute()':
astro.cc:8: undefined reference to `vtable for Astronaute'
/usr/bin/ld: /tmp/ccMLpjbY.o:(.data.rel.ro._ZTI10Cosmonaute[_ZTI10Cosmonaute]+0x10): undefined reference to
`typeinfo for Astronaute'
collect2: error: ld returned 1 exit status
```

Déroulement du programme (une fois les erreurs de compilation et d'édition de liens corrigées) :

```
double free or corruption
Segmentation fault (core dumped)
```

Réponses / explications :



```
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:
11    Astronaute(string nom_): nom{nom_} {}
12    virtual string nationalite() const;
13    const string& getNom() { return nom; }
14  };
15
16  class Cosmonaute: public Astronaute {
17    using Astronaute::Astronaute;
18    string nationalite() override {
19      return "Russe"; }
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_; }
28    virtual void decoller(Meteo) = 0;
29  };
30
31  ostream& operator<<(ostream& o, Vol& v) {
32    o << "Vol[a1=" << v.a1.getNom()
33      << ",a2=" << v.a2.getNom() << "]";
34  }
35
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        }
62      }
63      return true;
64    }
65  };
66
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  }
```

Réponses / explications :

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 :

Ne pas écrire dans cette zone.

```
#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;
}
```

Place supplémentaire pour répondre à n'importe quelle question si nécessaire. Mais VEUILLEZ INDIQUER LE NUMÉRO DE LA QUESTION TRAITÉE.