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 Système, sections IN et SC
Chappelier J.-C.

Programmation Orientée Objet (SMA/SPH) :

CORRIGÉ DE LA
SÉRIE NOTÉE

20 avril 2023

INSTRUCTIONS (à lire attentivement)

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

1. Vous disposez de 1h45 pour faire cette série notée (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 ; utilisez aussi le verso des feuilles, **MAIS** n'utilisez *que* le verso de la feuille sur laquelle se trouve la question, et non **pas** celui de la feuille précédente !
   Ne joignez aucune feuille supplémentaire ; **seul ce document sera corrigé**.
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. Cette série notée comporte deux exercices indépendants (pages 2 et 3), 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 98). Les deux exercices comptent pour la note finale.

Exercice 1 – Solutions chimiques [sur 12 points]

Voici une correction possible :

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

class Flacon
{
private:
    string nom;
    double volume;
    double pH;

public:
    Flacon(const string& unNom, double unVolume, double unPh)
        : nom(unNom), volume(unVolume), pH(unPh)
    {}

    Flacon operator+(Flacon f2) const
    {
        f2.pH = -log10( (      volume * pow(10.0,    -pH)
                        + f2.volume * pow(10.0, -f2.pH))
                    / (volume  + f2.volume)
                  );

        f2.volume += volume;
        f2.nom = nom + " + " + f2.nom;
        return f2;
    }

    void affiche(ostream& sortie) const
    {
        sortie << nom << " : " << volume << " ml, " << "pH " << pH;
    }
};

// ====================================================================
ostream& operator<<(ostream& sortie, Flacon const& f)
{
    f.affiche(sortie);
    return sortie;
}
```

On aurait aussi pu définir `Flacon::operator+=()` et faire que `Flacon::operator+()` l'appelle.

Exercice 2 – Roulette [sur 86 points]

On s'intéresse ici à simuler des stratégies de jeu pour le jeu de la roulette. On vous demande pour cela d'écrire un programme en C++, en utilisant une approche « orientée objets » avec le moins de duplication de code possible.

Pour déjà donner une idée générale, voici un exemple de `main()` possible et le résultat attendu correspondant. Regardez le bien avant de commencer à coder et revenez y autant que nécessaire lors de la lecture de la suite de la donnée.
Pour que ce soit plus pratique, cet exemple est redonné en toute dernière dernière page.

```
int main() {
  Joueur M("Mortimer");
  // miser 100 jetons sur le 1
  Pleine p1(100, 1);
  M.set_strategie(p1);

  Joueur B("Blake");
  // miser 30 jetons sur les rouges
  Rouges p2(30);
  B.set_strategie(p2);

  Jeu j;
  j.participe(B);
  j.participe(M);

  for (auto tirage : { 12, 1, 31 }) {
    j.rien_ne_va_plus(tirage);
    j.affiche_resultats();
    cout << endl;
  }

  // miser 50 jetons sur le 17
  Pleine p3(50, 17);
  M.set_strategie(p3);
  j.rien_ne_va_plus(17);
  j.affiche_resultats();
  cout << endl;
}
```

```
Mortimer choisit la stratégie : mise 100 sur le 1
Blake choisit la stratégie : mise 30 sur les rouges
Croupier : le numéro tiré est le 12
Le joueur Blake mise 30 sur les rouges et perd sa mise
Le joueur Mortimer mise 100 sur le 1 et perd sa mise

Croupier : le numéro tiré est le 1
Le joueur Blake mise 30 sur les rouges et gagne 30
Le joueur Mortimer mise 100 sur le 1 et gagne 3500

Croupier : le numéro tiré est le 31
Le joueur Blake mise 30 sur les rouges et perd sa mise
Le joueur Mortimer mise 100 sur le 1 et perd sa mise

Mortimer change de stratégie : mise 50 sur le 17
Croupier : le numéro tiré est le 17
Le joueur Blake mise 30 sur les rouges et perd sa mise
Le joueur Mortimer mise 50 sur le 17 et gagne 1750
```

La roulette est un jeu de hasard dans lequel chaque joueur mise [1] des jetons sur un ensemble de numéros de couleur rouge ou noire, compris entre 0 et 36 (sans couleur pour le 0).

Quand les joueurs ont effectué leur mise, le croupier lance la bille et annonce « rien ne va plus » [2]. À partir de ce moment, plus aucune mise n'est admise. Quand la bille s'arrête, les joueurs reçoivent leur gain [3] ou perdent leur mise.

Les règles de la roulette déterminent les types de mises qui sont possibles. Chaque type de mises définit un sous-ensemble particulier de numéros, ainsi que le gain obtenu si le numéro tiré appartient à l'ensemble. Par exemple, une mise sur les rouges est gagnante si le numéro tiré est rouge ; elle rapporte alors une fois la somme misée.

1. "*bet*" in English. 2. "*no more bets*" in English. 3. "*winnings*" in English.

Un joueur qui effectue une mise gagnante remporte le gain correspondant et conserve la somme misée. Toutes les mises portant sur des ensembles de numéros ne contenant pas le numéro tiré sont perdues par les joueurs.

Question 2.1 – Les mises [sur 42 points]

Question 2.1.1 – Les mises générales [sur 17.5 points]

Une mise comprend un nombre de jetons misés, ainsi qu'un moyen de calculer le gain en fonction du numéro tiré et un moyen permettant d'afficher la mise[4]. Une fois fixé, le nombre de jetons misés ne peut être changé. Par ailleurs, ces deux fonctionnalités (calcul de gain et affichage) doivent pouvoir être spécifiques à chaque type de mises (voir plus loin).

Définissez une classe `Mise` correspondant la description ci-dessus. Cette classe est prévue pour être une description générale et ne devra pas pouvoir être instanciée. Elle devra par ailleurs avoir un moyen d'accéder au nombre de jetons misés et pouvoir être affichée au moyen de l'opérateur usuel `<<`.
Par défaut, une `Mise` s'affiche simplement en affichant seulement « *mise* » suivi du nombre de jetons misés ; p.ex. : `mise 100`.

Vous êtes libre de concevoir cette classe comme vous le souhaitez, tant qu'elle vérifie les contraintes ci-dessus et les principes de bonne programmation. Si un comportement ou une situation donnée n'est pas définie dans la consigne, vous êtes libre de choisir le comportement adéquat.
Cette remarque s'applique également à tout le reste de votre code.

Réponse : (vous pouvez répondre en deux colonnes si nécessaire, et avez aussi un peu de place au dos.)

```
typedef unsigned int Jeton; // optional
typedef unsigned int Tirage; // optional

class Mise {
public:
  Mise(Jeton m) : mise_(m) {}

  // optional
  virtual ~Mise() {}
  // no real nead to copy nor move Mise

  Jeton mise() const { return mise_; }
  virtual Jeton gain(Tirage) const = 0;
  virtual void affiche(ostream& out) const
  { out << "mise " << mise_ ; }

private:
  const Jeton mise_;
};

ostream& operator<<(ostream& out, const Mise& m)
{
  m.affiche(out);
  return out;
}
```

4. Rappel: si nécessaire, voir l'exemple de déroulement fourni en début de donnée (et reproduit en dernière page).

Question 2.1.2 – Les mises spécifiques [sur 24.5 points]

Pour simplifier, nous allons supposer que seuls les deux types de mises suivantes sont possibles (en réalité, de nombreux autres types de mises sont possibles, mais nous ne les représenterons pas ici) :

- les mises portant sur un seul numéro ; ces mises rapportent 35 fois le nombre de jetons misés ;
- la mise sur les rouges, laquelle rapporte une fois le nombre de jetons misés ;
  les numéros rouges sont 1, 3, 5, 7, 9, 12, 14, 16, 18, 19, 21, 23, 25, 27, 30, 32, 34 et 36 (dans votre code, vous pouvez raccourcir cette liste avec « ... »).

Créez (et définissez *complètement*) une classe `Pleine` pour représenter les mises sur un seul numéro et une classe `Rouges` pour représenter les mises sur les rouges. Les mises `Pleine` s'affichent au format suivant [5] :

`mise <nb jetons> sur le <numéro>`

et les mises `Rouge` au format suivant :

`mise <nb jetons> sur les rouges`

Réponses :

```
class Pleine : public Mise {
public:
  Pleine(Jeton m, Tirage c) : Mise(m), case_(c) {
    // this check would better be a global function, like check(c)
    if ((c < 0) or (c > 36)) { // c < 0 is useless if unsigned
      // do something: throw or at least cout <<
    }
  }

  virtual Jeton gain(Tirage) const override;

  virtual void affiche(ostream& out) const
  {
    Mise::affiche(out);
    out << " sur le " << case_;
  }

private:
  Tirage case_;

  // bonus
  static constexpr Jeton ratio = 35;
};

// in this exam, can also be inside the class
Jeton Pleine::gain(Tirage c) const {
  if (c == case_) {
    return ratio * mise();
  } else {
    return 0;
  }
}
```

5. Rappel: si nécessaire, voir l'exemple de déroulement fourni en début de donnée (et reproduit en dernière page).

```
class Rouges : public Mise {
  public:
  using Mise::Mise; // or any other valid similar
  virtual Jeton gain(Tirage) const override;

  virtual void affiche(ostream& out) const
  {
    Mise::affiche(out);
    out << " sur les rouges";
  }

private:
  // they don't know 'set' yet
  static constexpr array<Tirage, 18> rouges = {1,3,5,...,36};
};

// in this exam, can also be inside the class
Jeton Rouges::gain(Jeton c) const {
  for (auto r : rouges) {
    if (c == r) { return mise(); }
  }
  return 0;
}
```

Question 2.2 – Les joueurs [sur 23 points]

On représentera les joueurs d'une partie de roulette par une classe `Joueur` qui possède un nom (non modifiable) et une stratégie de mise. Cette stratégie de mise utilisée par le joueur pourrait changer au moyen d'une méthode `set_stratégie()` [6].

Lorsque `set_stratégie()` est appelée, un message sera affiché (sur `cout`) indiquant la mise choisie. Ce message commencera par la nom du joueur et « ***choisit la*** *stratégie* » si c'est la première fois ou « ***change de*** *stratégie* » s'il en change. Voir des exemples dans l'exemple de déroulement fourni en début de donnée.

Par ailleurs, la stratégie de mise, ainsi que le nom du joueur pourront être obtenus grace à des « méthodes *get* ».
Indication : attention cependant aux « fuites d'encapsulation ».

Réponse :

```
class Joueur {
public:
  Joueur(const string& n) : nom_(n), strategie_(nullptr) {}
  string nom() const { return nom_; }
  void set_strategie(Mise const& s);
  const Mise& strategie() const { return *strategie_; } // ATTENTION aux fuites
                                                        // d'encapsulation ici !!
private:
  const string nom_;
  const Mise* strategie_; // « cas numéro 1 » : référence
};

// in this exam, can also be inside the class
void Joueur::set_strategie(Mise const& s)
{
  cout << nom_ << " ";
  if (strategie_ == nullptr)
    cout << "choisit la";
  else
    cout << "change de";
  cout << " stratégie : " << s << endl;

  strategie_ = &s;
}
```

Le pointeur sur `Mise` doit être un pointeur car on ne peut pas avoir d'instance de `Mise` (classe abstraite), donc pas de copie, et ce ne peut être qu'un pointeur à la C, car il n'y a pas d'allocation dynamique (cf `main()`) et on ne peut pas réaffecter une référence.

Au niveau du « getter » de stratégie : ce n'est pas forcément la meilleure option. Il ne faut en tout cas pas faire de « fuite d'encapsulation » ici, donc pas retourner de pointeur ; et on ne peut pas non plus retourner de copie, puisque `Mise` est une classe abstraite !
Une meilleure solution serait ne de pas du tout avoir ce « getter » et offrir une surcharge de l'affichage des `Joueur`, ainsi qu'une méthode `gain()` (qui retourne simplement le gain de sa stratégie).

6. Rappel: si nécessaire, voir l'exemple de déroulement fourni en début de donnée (et reproduit en dernière page).

Question 2.3 – Le jeu [sur 21 points]

Pour représenter une partie de roulette, créez une classe `Jeu` comprenant (au moins) :

- un ensemble de joueurs ;
- le dernier numéro tiré ;
- une méthode `participe()` pour y ajouter un joueur ;
- une méthode `rien_ne_va_plus()` qui fixe le numéro tiré ;
  pour tester notre jeu, nous tricherons ici et la méthode `rien_ne_va_plus()` recevra en argument le numéro que l'on veut voir sortir et qui sera donc le numéro effectivement tiré [7] ;
- une méthode `affiche_resultats()` qui affiche sur le terminal le déroulement d'un tirage comme dans l'exemple fourni en début de donnée.

Réponse :

7. Rappel: si nécessaire, voir l'exemple de déroulement fourni en début de donnée (et reproduit en dernière page).

```
class Jeu {
public:
  Jeu() = default; // remis, cause suppr. constr. copie (BONUS)

  void rien_ne_va_plus(int valeur) // better: unsigned
  {
    // this check would better be a global function, like check(valeur)
    if ((valeur < 0) or (valeur > 36)) {
      // do something: throw or at least cout <<
    } else {
      tirage = valeur;
    }
  }

  void participe(Joueur const& p) {
      /* Note : on pourrait ajouter une vérification que le joueur
       * ne participe pas déjà...
       * (ils ne connaissent pas encore les set)
       */
      participants.push_back(&p);
    }
    void affiche_resultats() const;

  private:
    Jeu(const Jeu&) = delete; // BONUS : interdiction copie
    Jeu& operator=(const Jeu&) = delete;

    vector<const Joueur*> participants;
    int tirage;
};

void Jeu::affiche_resultats() const
{
  cout << "Croupier : le numéro tiré est le " << tirage << endl;
  for (auto joueur : participants) {
    cout << "Le joueur " << joueur->nom() << " "
        << joueur->strategie() << " et ";
    const Jeton gain(joueur->strategie().gain(tirage));
    if (gain == 0) {
      cout << "perd sa mise";
    } else if (gain > 0) {
      cout << "gagne " << gain;
    }
    cout << endl;
  }
}
```

Là aussi, les participants au `Jeu` ne peuvent être que des pointeurs à la C car :

- il ne peuvent pas être des `Joueur` directement (la copie ne fait pas de sens ici ; le `Jeu` fait clairement *référence* aux joueurs, existants par ailleurs (cf le `main()`)) ;
- on ne peut pas avoir de collection de références ;
- il n'y a aucune raison d'avoir des « smart pointers ».