PROG2 : Programmation avancée et C++

Examen du 11 mai 2021 (2 h)

Tous documents interdits à l'exception d'un A4 recto verso manuscrit de votre main. Calculatrices, téléphones, ordinateurs et montres connectées interdites. La correction tiendra compte de la qualité de l'argumentaire et de la présentation. Un code illisible et/ou incompréhensible est un code faux.

★ **Exercice 1 :** Classe `Fraction` (2pts).

Écrivez une classe permettant de représenter une fraction entière **constante**. Vous doterez votre classe des accesseurs `numerateur()` et `denominateur()` ainsi que des opérateurs permettant de calculer la somme et le produit de deux fractions et celui permettant d'afficher une fraction sous forme textuelle.

★ **Exercice 2 :** Surcharge et redéfinition de méthodes (5pts).

On compile et exécute le programme ci-dessous avec `g++` avec les paramètres `-Wall -Wextra`. Comme `-Wsuggest-override` n'est pas utilisé, les `override` manquants ne posent aucun problème.

```
1  #include <iostream>
2  using namespace std; // Abrège std::cout en cout
3
4  class Up {
5  public:
6    void f() { cout << "Up::f()" << endl; }
7    virtual void g() { cout << "Up::g()" << endl; }
8  };
9
10 class Down : public Up {
11 public:
12   void f() { cout << "Down::f()" << endl; }
13   void f(int i) { cout << "Down::f("<<i<<")" << endl; }
14   virtual void g() { cout << "Down::g()" << endl; }
15 };
```

```
17 int main() {
18   Up uu;
19   Down dd;
20   Up* ud = new Down();
21   uu.f();
22   uu.f(1);
23   uu.g();
24   dd.f();
25   dd.f(1);
26   dd.g();
27   ud->f();
28   ud->f(1);
29   ud->g();
30 }
```

▷ **Question 1 :** Indiquez le type statique et dynamique de chaque variable `uu`, `dd` et `ud`.

▷ **Question 2 :** Pour chacune des lignes 21 à 29, indiquez (en justifiant très brièvement vos réponses) :

1. si elle compile ou non, en explicitant les éventuelles erreurs et warning produits ;
2. le message affiché à l'exécution (si la compilation est possible).

▷ **Question 3 :** Relevez un exemple de surcharge de méthode et un exemple de redéfinition de méthode dans le code ci-dessus.

▷ **Question 4 :** Expliquez le fonctionnement des différents mécanismes de sélection du code à exécuter (*code binding*) en C++. La réponse attendue est de l'ordre de la dizaine de lignes d'explications, éventuellement accompagnée de schémas explicatifs.

★ **Exercice 3 :** Design de code (6pts).

Le jeu traditionnel indien *échelles et serpents* ressemble un peu au jeu de l'oie. Chaque joueur à son tour lance un dé, puis avance son pion du nombre de cases indiqué. Les cases *échelle* font avancer le pion vers la case indiquée, tandis que les cases *serpent* font reculer le joueur. Le premier joueur à arriver sur la case 100 gagne. C'est donc un pur jeu de chance.

Le code `snake_ladder.cpp` donné en annexe est une implémentation presque complète de ce jeu. Seule la fonction `waitkey()` (qui attend qu'une touche soit pressée) est omise, ainsi que les entêtes.

▷ **Question 1 :** (prise en main) Expliquez les lignes 27 à 32.

La qualité du code proposé pose problème, tant au niveau des traitements effectués qu'au niveau des données manipulées. L'objectif de cet exercice est de proposer des pistes d'amélioration.

▷ **Question 2 :** Proposez un redécoupage de ce code en plusieurs fonctions. Indiquez le prototype de chaque fonction ainsi que les lignes du code original correspondantes, mais il n'est pas nécessaire d'écrire le corps de chaque fonction. Expliquez simplement les éventuelles modifications à apporter au code repris.

▷ **Question 3 :** On souhaite maintenant permettre à un nombre variable de joueurs de jouer. Pour plus de convivialité, on demandera le nom de chaque joueur en début de partie. On souhaite également enregistrer les scores d'une partie sur l'autre pour organiser des championnats en plusieurs manches.

Quelles sont les modifications à apporter au programme ? Ici encore, vous n'avez pas à écrire l'intégralité du code nécessaire, mais vous devez nous faire comprendre l'intention du code que vous écririez.

▷ **Question 4 :** On souhaite maintenant ajouter une nouvelle interface (par exemple basée sur SFML) en plus de l'existante. Plus précisément, on souhaite faire un moteur de jeu générique qui puisse être utilisé avec l'une ou l'autre des interfaces. Proposez un découpage en plusieurs classes, en listant les responsabilités et collaborateurs de chaque classe. Vous n'avez pas à écrire de code à cette question.

★ **Exercice 4 :** Comprendre du code C++ (reprise du TP3 – 7pts)

Observez le code `mystery_picture.cpp` donné en annexe. Ce code est complet, à l'exception des entêtes et de la fonction `draw()` (qui dessine à l'écran une figure), omises du listing présenté. Il compile et s'exécute sans erreur.

Ce code utilise la bibliothèque graphique SFML. Dans ce système, le point $(0, 0)$ se trouve dans l'angle en haut à gauche et les rotations se font dans le sens des aiguilles d'une montre. Les couleurs utilisées sur la figure ci-dessous sont nommées `sf::Color::Green` (haut gauche), `sf::Color::Red` (haut droite), `sf::Color::Blue` (bas gauche) et `sf::Color::Magenta` (bas droite).

▷ **Question 1 :** Dessinez un diagramme des classes fournies indiquant les relations d'héritage et celles de composition. Vous préciserez pour chaque classe si elle est abstraite ou concrète. Ajoutez une légende à votre schéma pour spécifier les notations choisies.

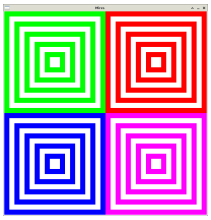
▷ **Question 2 :** Que dessine la figure `pic`, définie à partir de la ligne 80 ?

▷ **Question 3 :** Donnez le schéma mémoire correspondant à cette structure de données `pic`. Vous représenterez tous les objets en mémoire (en indiquant s'ils sont sur le tas, la pile ou ailleurs), ainsi que les pointeurs des uns vers les autres.

▷ **Question 4 :** Généralisez la classe `Above` pour prendre un nombre arbitraire de sous-figure. Écrivez une classe `Beside` similaire à la classe `Above`, mais horizontalement, ainsi qu'une classe `OnTop` qui fait une superposition de figures sans les modifier. Factorisez le code autant que possible.

▷ **Question 5 :** Écrivez une classe `Scaled` permettant de changer l'échelle d'une figure. Cette classe doit avoir le constructeur suivant : `Scaled(Picture* p, double ratio)`. Factorisez le code autant que possible.

▷ **Question 6 :** Écrivez le code nécessaire pour créer la figure de droite. Elle est constituée de quatre mires de différentes couleurs. Chaque mire est une superposition de boîtes de plus en plus petites. Par exemple, la mire verte est une succession de boîtes vertes puis blanches dont la taille réduit de 10% à chaque fois.



Vous êtes libre d'ajouter une ou plusieurs classes au projet si cela améliore la lisibilité de votre code.

snake_ladder.cpp

```
int main() {
  int player1 = 1, player2 = 1;
  srand(time(nullptr)); // Initialize the pseudo-random

  // Some cells teleport the player to another cell.
  // mover[i]=j means that when you arrive to the cell i, you're teleported to cell j
  std::vector<int> mover;
  for (int i = 0; i <= 106; i++)
    mover.push_back(i);
  for (auto elem : std::vector<std::vector<int>> {
      {98,28}, {95,24}, {92,51}, {83,19}, {69,33}, {64,36}, {48,9},{8,26},{55,7},{73,1},{46,5},
      {21,82}, {43,77}, {50,91}, {54,93}, {62,96}, {66,87},{44,22},{52,11}, {59,17},{80,100} })

    mover[elem[0]] = mover[elem[1]];

  std::cout << "============================================================\n\n";
  std::cout << "\t\tSNAKE LADDER GAME\n\n";
  std::cout << "============================================================\n\n";

  int turn = 1;
  while (player1 < 100 && player2 < 100) {

    std::cout << "----------------[ turn " << turn++ << " ]------------------------------\n";
    for (int cell = 1; cell <= 100; cell++) {
      std::string content;
      if (player1 == cell)
        content = "*P1* ";
      if (player2 == cell)
        content += "*P2* ";
      if (mover[cell] > cell)
        content += "ladder to " + std::to_string(mover[cell]);
      if (mover[cell] < cell)
        content += "snake to " + std::to_string(mover[cell]);
      std::cout << std::setw(3) << cell << "[" << std::setw(13) << content << "]";
      if (cell % 10 == 0) // Print 10 cells on each line
        std::cout << std::endl;
    }
    std::cout << "------------------------------------------------------------\n";
    std::cout << "\n---> Player 1: press a key to roll the dice... ";
    waitkey();
    int lastposition = player1;
    int dice         = random() % 6 + 1;
    player1          = mover[player1 + dice];
    std::cout << "You rolled a " << dice << "!! Now you are at position " << player1;
    if (player1 < lastposition)
      std::cout << ". Oops!! Snake found !!";
    else if (player1 > lastposition + 6)
      std::cout << ". Great!! you got a ladder !!";

    std::cout << "\n\n---> Player 2: press a key to roll the dice... ";
    waitkey();
    lastposition = player2;
    dice         = random() % 6 + 1;
    player2      = mover[player2 + dice];
    std::cout << "You rolled a " << dice << "!! Now you are at position " << player2;
    if (player2 < lastposition)
      std::cout << ". Oops!! Snake found !!";
    else if (player2 > lastposition + 6)
      std::cout << ". Great!! you got a ladder !!";
    std::cout << "\n\n(press a key to start the next turn)\n\n";
    waitkey();
  }
  std::cout << "\n\n++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++\n";
  if (player1 >= player2)
    std::cout << "\t\tPlayer 1 wins the game\n";
  else
    std::cout << "\t\tPlayer 2 wins the game\n";
  std::cout << "++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++\n\n";
  return 0;
}
```

mystery_picture.cpp

```
const int winsize = 800; // in pixels

class Picture {
public:
  virtual void draw(sf::RenderWindow*, sf::Transform trans) = 0;
};

class Triangle : public Picture {
  sf::ConvexShape shape;
public:
  Triangle(sf::Color color = sf::Color::Green) {
    shape.setPointCount(3);
    shape.setPoint(0, sf::Vector2f(0.f, winsize));
    shape.setPoint(1, sf::Vector2f(winsize, winsize));
    shape.setPoint(2, sf::Vector2f(winsize, 0));
    shape.setFillColor(color);
  }
  void draw(sf::RenderWindow* win, sf::Transform trans) { win->draw(shape, trans); }
};

class Box : public Picture {
  sf::RectangleShape shape{sf::Vector2f(winsize, winsize)};
public:
  Box(sf::Color c = sf::Color::Red) { shape.setFillColor(c); }
  void draw(sf::RenderWindow* win, sf::Transform trans) { win->draw(shape, trans); }
};

class FourPics : public Picture {
  std::vector<std::shared_ptr<Picture>> pics_;
public:
  FourPics(std::shared_ptr<Picture> p1, std::shared_ptr<Picture> p2,
           std::shared_ptr<Picture> p3, std::shared_ptr<Picture> p4) : pics_({p1,p2,p3,p4}) {}
  void draw(sf::RenderWindow* win, sf::Transform trans) {
    auto half = sf::Vector2f(.5, .5);
    pics_[0]->draw(win, sf::Transform(trans).translate(winsize / 2, 0).scale(half));
    pics_[1]->draw(win, sf::Transform(trans).translate(0, 0).scale(half));
    pics_[2]->draw(win, sf::Transform(trans).translate(0, winsize / 2).scale(half));
    pics_[3]->draw(win, sf::Transform(trans).translate(winsize / 2, winsize / 2).scale(half));
  }
};
class Above : public Picture {
  std::shared_ptr<Picture> p1_;
  std::shared_ptr<Picture> p2_;
public:
  Above(std::shared_ptr<Picture> p1, std::shared_ptr<Picture> p2) : p1_(p1), p2_(p2) {}
  void draw(sf::RenderWindow* win, sf::Transform trans) {
    p1_->draw(win, sf::Transform(trans).translate(0, 0).scale(1, 0.5));
    p2_->draw(win, sf::Transform(trans).translate(0, winsize/2).scale(1, 0.5));
  }
};

class Rotate : public Picture {
  std::shared_ptr<Picture> p_;
  int angle_; // Clock-wise
public:
  Rotate(std::shared_ptr<Picture> p, int angle) : p_(p), angle_(angle) {}
  void draw(sf::RenderWindow* win, sf::Transform trans) {
    p_->draw(win, sf::Transform(trans).rotate(angle_, winsize / 2, winsize / 2));
  }
};

int main(int, char**)
{
  auto triB = std::make_shared<Triangle>(sf::Color::Black);
  auto triR = std::make_shared<Triangle>(sf::Color::Red);
  auto pic = std::make_shared<Above>
    (std::make_shared<FourPics>(triB, triB, triR, triR),
     std::make_shared<Above>(std::make_shared<Box>(sf::Color::White),
                             std::make_shared<Rotate>(triR,90)));
  draw(pic, "Mystery");

  return 0;
}
```