# POO & C++ : Examen sur papier

EISE4 2014—2015

Pablo Rauzy

rauzy@enst·fr

pablo.rauzy.name/teaching.html#epu-cpp

21 novembre 2014

Pour les questions types QCM, plusieurs choix peuvent être justes. Dans ce cas ils doivent tous être cochés.

**1.** C++ est un langage :
☐ orienté objet ☐ fonctionnel ☐ impératif ☐ statiquement typé ☐ dynamiquement typé

**2.** `Shape *fig;`
En supposant que le code ci-dessus est valide, on dit que `fig` est _____ de _____ `Shape`.
☐ une variable / la classe ☐ une variable / l'instance ☐ une instance / la classe

**3.** `Shape *fig = new Square(42);`
Ici, on dit que le _____ de `fig` est `Square *` et que son _____ est `Shape *`.
☐ type statique / type dynamique ☐ type dynamique / type statique

**4.** Le code de la question précédente implique que _____ dérive de _____.
☐ `Shape` / `Square` ☐ `Square` / `Shape`

**5.** Si la classe D dérive de la classe B, on dit que B est _____ de D.
☐ un sous-objet ☐ un sous-type ☐ une sous-classe

**6.** Si A et B dérivent de X, C dérive de Y, et D dérive de X et Y, et E dérive de A. Si une fonction attend en argument un objet de type X, on peut lui passer un objet de type :
☐ A ☐ B ☐ C ☐ D ☐ E ☐ X ☐ Y

**7.** Par rapport à `malloc` et `free`, que font `new` et `delete` en plus ?

**8.** Quelle syntaxe permet de modifier la cible d'un pointeur `p` ?
☐ `*p = a;` ☐ `p = a;` ☐ `&p = a;` ☐ c'est impossible

**9.** Quelle syntaxe permet de modifier la cible d'une référence `r` ?
☐ `*r = a;` ☐ `r = a;` ☐ `&r = a;` ☐ c'est impossible

**10.** Quel écueil le mot clef `const` permet d'éviter lorsqu'il qualifie un type référence en argument d'une fonction ?

**11.** Dans une classe, les constructeurs de ses objets membres sont appelés _____ le code de son constructeur.
☐ avant ☐ après ☐ ça dépend du compilateur

**12.** Dans une classe, les destructeurs de ses objets membres sont appelés _____ le code de son destructeur.
☐ avant ☐ après ☐ ça dépend du compilateur

**13.** Les constructeurs des classes de base directes d'une classe sont appelés _____ ceux des ses objets membres.
☐ avant ☐ après ☐ ça dépend du compilateur

**14.** Les destructeurs des classes de base directes d'une classe sont appelés _____ ceux de ses objets membres.
☐ avant ☐ après ☐ ça dépend du compilateur

**15.** Quelle est la différence entre une `class` et une `struct` ?

**16.** Quelle est la différence entre l'opérateur `.` et l'opérateur `->` pour l'accès aux membres d'une instance ?

**17.** Lorsque certaines données membres sont ou contiennent des pointeurs, à quoi faut-il faire attention lors de la copie d'une instance ?

**18.** Qu'implique le mot clef `static` dans la déclaration d'une fonction membre pour celle-ci ?

**19.** À quoi sert le mot clef `explicit` devant un constructeur ?

**20.** Si dans la déclaration d'une classe A on trouve la ligne `friend class B;`, cela veut dire que :
☐ les fonctions membres de la classe B peuvent accéder aux membres privés de la classe A
☐ les fonctions membres de la classe A peuvent accéder aux membres privés de la classe B

**21.** L'héritage permet :
☐ la généricité ☐ le polymorphisme

**22.** Les modèles (templates) permettent :
☐ la généricité ☐ le polymorphisme

**23.** Un modèle peut prendre comme paramètre :
☐ n'importe quel type ☐ seulement des types classes ou structures ☐ seulement des types de base
☐ n'importe quelle valeur ☐ des valeurs connues à la compilation ☐ des valeurs connues à l'exécution

**24.** Lors de l'utilisation du polymorphisme, on accède à un objet via son interface _____.
☐ statique ☐ dynamique

**25.** Expliquez alors l'intérêt de déclarer des fonctions membres comme étant virtuelles.

**26.** Dans le cas du polymorphisme, quelle(s) fonction(s) membre(s) en particulier il est important de déclarer comme étant virtuelle(s) ?
☐ le constructeur ☐ le destructeur ☐ toutes celles qui modifient l'état de l'objet

**27.** Une fonction virtuelle n'ayant pas d'implémentation est :
☐ abstraite ☐ virtuelle pure ☐ c'est impossible

**28.** Une classe possédant une ou plusieurs telles fonctions est :
☐ abstraite ☐ virtuelle pure ☐ instanciable ☐ non-instanciable ☐ c'est impossible

**29.** Lorsqu'on surcharge une fonction, on peut modifier :
☐ le type de sa valeur de retour ☐ son nom ☐ les types de ses arguments ☐ son arité

**30.** Lorsqu'on surcharge un opérateur, on peut modifier :
☐ sa priorité ☐ son associativité ☐ son arité ☐ sa commutativité

**31.** Quel est l'intérêt des différentes politiques d'accès aux membres d'une classe ? Donnez une exemple d'utilisation de chaque.

**32.** Quel est l'intérêt des différentes politiques de dérivation d'une classe ? Donnez une exemple d'utilisation de chaque.

**33.** On souhaite utiliser un conteneur de la STL, quel est le principal critère de choix pour choisir lequel ? Donnez deux exemples de choix différents en les justifiant par rapport à ce critère.

**34.** Dans la STL, quel rôle joue les itérateurs pour les conteneurs, par rapport à un tableau "classique" (comme en C par exemple) ?

**35.** Corrigez les erreurs présentent dans le code suivant :

```
1  class Shape {
2    explicit Shape (int pos_x, int pos_y);
3    ~Shape ();
4    int getPositionX ();
5    int getPositionY ();
6    void move (int x, int y);
7    virtual void draw () = 0
8  private:
9    int position[2];
10 }
```

Considérez le code suivant :

```
class A {
public:
  A () { cout << "Cons A; "; }
  A (const A &a) { cout << "Copy A; "; }
  virtual ~A () { cout << "Destr A; "; }
};
class B {
public:
  B () { cout << "Cons B; "; }
  B (const B &b) { cout << "Copy B; "; }
  ~B () { cout << "Destr B; "; }
};
class X : public A {
public:
  X () { cout << "Cons X; "; }
  X (const X &x) { cout << "Copy X; "; }
  ~X () { cout << "Destr X; "; }
};
class Y : public B {
public:
  Y () { cout << "Cons Y; "; }
  Y (const Y &x) { cout << "Copy Y; "; }
  ~Y () { cout << "Destr Y; "; }
};
void f (A a) { cout << "f(A); "; }
void f (A *a) { cout << "f(A *); "; }
void f (B b) { cout << "f(B); "; }
void f (B *b) { cout << "f(B *); "; }
```

Pour les question 36 à 43, écrivez la sortie du code donné.

**36.** `A a; f(a);`

**37.** `A a; f(&a);`

**38.** `X x; f(x);`

**39.** `X x; f(&x);`

**40.** `X *x = new X; f(x); delete x;`

**41.** `A *a = new X; f(a); delete a;`

**42.** `Y *y = new Y; f(y); delete y;`

**43.** `B *b = new Y; f(b); delete b;`

**44.** Comment s'appelle le créateur du langage C++ ?