POUPA 1/2
Adrien
L'3 2018
Groupe C

Devoir écrit C++ (II)

25/11/2015

17,50
Excellent travail

POUPA Adrien
L3PRIME 2015

4,5/5

Questions de cours

1) Un attribut de classe est partagé par toutes les instances de classe. De plus, contrairement à un attribut d'instance, il est accessible sans même que la classe soit instanciée. 0,5

2) Une méthode de classe est une fonction dans une classe ayant accès à tous les attributs de la classe, par exemple le constructeur. 0

3) L'opérateur << doit être surchargé par une fonction amie, puisque faisant partie de la librairie iostream. Il ne peut donc pas être surchargé par une surcharge interne à proprement parler, mais il peut faire appel à une méthode de la classe. 0,5

4) La "liaison dynamique", dite aussi "polymorphisme" permet d'appliquer la méthode correspondant à la nature effective de l'objet. Elle est déterminée au moment de l'exécution du programme et non de la compilation. Si la classe B hérite de A et qu'on appelle une méthode présente 0,5

dans les 2 classes, le polymorphisme permet de s'assurer que la méthode appelée correspondra à la nature de l'objet. On achève ce mécanisme en rajoutant le mot-clé virtual dans le fichier header.

5) Une méthode virtuelle pure est une méthode qui n'a pas d'implémentation dans le fichier cpp. Dans le fichier header, on note
virtual void maFonction() = 0;

0.5

Une classe possédant au moins une méthode virtuelle pure est dite abstraite.
Elle ne peut pas être instanciée.

6) Si on ne déclare pas le destructeur comme méthode virtuelle, dans une classe destinée à être dérivée, on risque de voir le mauvais destructeur appelé lors d'un delete, et on risque de ne pas libérer la mémoire correctement, ce qui à terme peut entraîner un plantage de l'hôte du programme.

0.5

Bien

7) Une déclaration de type template <class T> peut induire la création de méthodes génériques fonctionnant avec plusieurs types. Par exemple, une méthode

```
template <class T>
T max(const T& a, const T& b) {
    return ((a<b) ? a : b);
}
```

0.5

pourra aussi bien fonctionner avec des int, des double, des float... Cela évite d'écrire du code redondant.

8) Le type générique d'une fonction ou d'une telle classe doit être compatible avec l'utilisation qu'on en fait dans le main ou tout autre fichier cpp afin que le compilateur puisse l'instancier.
Dans l'exemple précédent, si on avait des char à la fonction max, elle ne fera que renvoyer des adresses de pointeurs, ce qui n'est pas le comportement attendu.
0,5 Il peut aussi être nécessaire de préciser le type de données T envoyées.
Dans ce cas on fait max<double>(a,b).
Si a et b sont des double.

9) • vector permet de créer un tableau redimensionnable dynamiquement
• list permet de créer une liste doublement chaînée ; on peut insérer des éléments en tête, en queue de liste et les afficher avec un itérateur
(0,5) • map permet de créer des tableaux associatifs dont le type est déterminé à la déclaration, par exemple map<char, int> test; et test["premier"] = 0; . On peut de même le parcourir

10) Le mécanisme des exceptions permet d'"attraper" des erreurs et d'arrêter l'exécution du code suivant l'erreur en affichant un message d'erreur préalablement défini pour le développeur. On a :

0,5

```
try {
    // le code à analyser
    // on peut aussi jeter une erreur avec throw
}
catch (Exception& e) {
std::cout << "Erreur : " << e.what() << endl;
}
```

Exercice 1 — 9,5/10 — T. Bg

(I) - Fichier "intvalue.h"

```
#include "value.h"
#include <sstream>
String intToString(int i);
class IntValue : public Value {
    int _value;
public:
    IntValue(int value = 0) : _value(value) {};
    void setValue(int value) { _value = value; }
    virtual string getStringValue() const {
        return intToString(_value);
    }
    virtual double getDoubleValue() const {
        return (double) _value; // Cast de _value en double
    }
```

(2)

Poux A
Adrien 2/2
L3 2018
Groupe C

```
};

string int2string(int i) {
    ostringstream os;
    os << i;
    return os.str();
}
```

Fichier "boolvalue.h"

```
#include "value.h"
#include <sstream>

class BoolValue : public Value {
    bool _value;
public:
    BoolValue(bool value = true) : _value(value) {};
    void setValue(bool value) { _value = value; }
    virtual string getStringValue() const {
        if (_value == true)
            return "true";
        else
            return "false";
    }
    virtual double getDoubleValue() const {
        if (_value == true)
            return 1.0;
        else
            return 0.0;
    }
};
```

1,5

- Fichier "StringValue.h"

```
#include "Value.h"
[redacted]

class StringValue : public Value {
    string _value;
public:
    StringValue(string value = "null") : _value(value) {};
    void setValue(string value) { _value = value; }
    virtual string getStringValue() const {
        return _value;
    }
    virtual double getDoubleValue() const {
        return (double) _value.length();
        // Il m'apparait difficile de retourner un double
        // à partir d'une string... je renvoie donc sa longueur
    }
};
```

(2)

(II) Si on prend en compte les classes écrits précédemment, on aura à l'affichage :

(2)
- true
- 1093
- -4.987
- hello

Avant de terminer main, on doit écrire (+2)

```
delete it;
delete value_set;
```

(III) On rajoute dans "value.h" :

```
friend ostream& operator<<(ostream& os, Value& v const)
{
    os << v.getStringValue() << endl;
    return os;
}
```

+0,5

Exercice 2 — 3/5

On doit changer le droit d'accès du constructeur. Pour obtenir la création d'une nouvelle instance et/ou obtenir l'adresse ou la référence, on passe par une méthode statique getSingleton. +1

- Fichier "singleton.h"

```
class Singleton {
private:
    Singleton(int singleton) : _singleton(singleton) {};
public:
    static int _singleton = 0;
    Singleton& getSingleton() {
        if (_singleton == 0) // Pas encore appelée
            _singleton = 1; // Création de l'instance
        return *this;
    }
    ~Singleton() { _singleton = 0; }
};
```

(2)

Ici, au premier appel de la classe, l'attribut de classe _singleton passé de 0 à 1.
À chaque appel suivant, on ne modifie plus cet élément et on se contente de retourner la référence de l'instance.
À la destruction de l'instance, l'attribut statique repasse à 0.