NOM POUPA

Prénom Adrien

Promo L'3 2018

Date 18/11/2015

POUPA Adrien
L3PRIME - 2015

1/3 15,25/20

MATIÈRE Programmation C++ (1)

Questions de cours 4,25/5

1) Le compilateur génère un fichier exécutable (.exe) par l'appareil sur lequel on veut exécuter un programme. Ce fichier contient toutes les sources et les dépendances nécessaires au bon fonctionnement du programme, de façon optimisée pour un temps d'exécution plus rapide et une place occupée moins grande. L'emplacement de tous les fichiers nécessaire à la compilation, est donné par l'éditeur de liens (linker). !!

+0,5

2) Une référence est l'alias d'une variable, désigné par un &. Là où le pointeur désigne directement l'adresse de l'objet pointé, la référence ne fait que rediriger vers l'objet. Une référence se fait sur un objet existant et on peut avoir autant d'alias qu'on veut sur une variable. Un delete sur la variable originale suffit. On s'en sert notamment pour pouvoir modifier une variable dans une fonction sans avoir à faire un return mais aussi pour traiter une variable renommée comme si on traitait la variable originale.

0,5

3)

0

4) La déclaration d'une classe se fait dans un fichier header.h et la définition des méthodes dans un fichier .cpp. Le fichier .h (parfois .hpp) ont le même nom que le fichier .cpp, et on ne met qu'une classe par fichier header.

0.5

5) Surcharger une fonction désigne le fait de déclarer plusieurs fois la même fonction prenant des attributs différents. Par exemple :

```
int addition (const int a, const int b)        (1)
{
    return a+b;
}
```

0.5

et (2)

```
int addition (const int a, const int b, const int c)
{ return a+b+c;
}
```

② est une surcharge de ①. Si on appelle addition() avec 2 entiers, ① est appelée; si on appelle la même fonction avec 3 entiers, ② est appelée.

b) Si on peut dire "est un(e)", c'est une relation de composition, sinon si on peut dire "a un(e)" c'est une relation d'aggrégation. (!)

ex: Une voiture est un véhicule
Une voiture a un volant.

7) Si on fait un new (dans le constructeur par exemple), il faut faire un delete, qui est fait dans le destructeur. Exemple:

maClasse.h (pour simplifier, je mets le code dans le .h)

```
Classe A {
    int *_un_entier;
public:
    A(const int un_entier) {
        *_un_entier = new int;
        _un_entier = un_entier;
    }
    ~A() {
        delete _un_entier;
    }
};
```

8) Un attribut "protected" dans la classe A aura le même comportement qu'un attribut privé.
Si la classe B hérite de A, il héritera aussi de l'attribut protected de A, mais pas de ses attributs privés.
Au sein de B, il aura le même comportement qu'un attribut privé. Et ainsi de suite pour chaque héritage.

0.5

9) Une méthode const garantit qu'elle ne modifiera pas de variables. Si on reprend la classe A définie plus haut, on pourrait faire :

```
int A::getNombreEntier() const {
    return _un_entier;
}
```

0.5

10) Une surcharge d'opérateur sert à faire des opérations comme l'addition ou la saisie d'un objet de façon naturelle dans le code, sans recourir à des fonctions comme addition écrite précédemment mais simplement en écrivant objet monObjet = monObjet1 + monObjet2;
On peut soit directement surcharger des opérateurs comme +, - ... dans le .h et .cpp, soit passer par des fonctions amies (friend) pour surcharger des opérateurs externes comme >> ou <<.

On peut donc passer par des fonctions amies ou internes.

0.5

NOM Poupa

Prénom

Promo

Date

2/3

MATIÈRE

Exercice 1

8/8 + 1,5

1/

```
#include <iostream>
using namespace std;
class DispositifEcriture {
    double _prix;
    string _marque;
public:
    DispositifEcriture (double prix=0, string marque="Generique");
    string getMarque() const;
    void setMarque(string marque);
    void setPrix(double prix);
    double getPrix() const;
    void ecrit(const string& message);
};
```

Fichier "DispositifEcriture.h"

+3

```
#include <iostream>
#include "DispositifEcriture.h"
DispositifEcriture::DispositifEcriture(double prix, string marque){
    _marque = marque;
    _prix = prix;
}
string DispositifEcriture::getMarque() const {
    return _marque;
}
```

Fichier "DispositifEcriture.cpp"

```
void DispositifEcriture::setMarque(string marque) {
    _marque = marque;
}

void DispositifEcriture::setPrix(int prix) {
    _prix = prix;
}

double DispositifEcriture::getPrix() const {
    return _prix;
}

void DispositifEcriture::ecrit(const string& message) {
    std::cout << message << endl;
}
```

#Le constructeur surcharge le constructeur par défaut.

Conception de classe

Il suffit que DispositifEcritureMajuscule hérite de DispositifEcriture.

Fichier
DispositifEcritureMajuscule.h

```
#include <iostream>
#include "DispositifEcriture.h"

class DispositifEcritureMajuscule : public DispositifEcriture {
public:
    DispositifEcritureMajuscule(double prix=0, string marque="Générique");
    Void ecrit(const string& message);
};
```

Fichier
DispositifEcritureMajuscule.cpp

```
#include <iostream>
#include "DispositifEcritureMajuscule.h"

DispositifEcritureMajuscule::DispositifEcritureMajuscule(double prix,
string marque) : DispositifEcriture(prix, marque) {
// On aurait pu utiliser la syntaxe : _attribut(attribut) plus
// haut
}

Void DispositifEcritureMajuscule::ecrit(const string& message) {
    std::cout << message.toupper() << endl;
}
```

Fichier
main.cpp

```
#include <iostream>
#include "DispositifEcriture.h"
#include "DispositifEcritureMajuscule.h"

int main() {
    DispositifEcriture machine1;
    DispositifEcritureMajuscule machine2;
```

```
machine1.ecrit("message de test");
machine2.ecrit("message de test");

return 0;
}
```

À l'affichage :

message de test
MESSAGE DE TEST

Exercice 2

3|7

Fichier gint.h

```
#include <iostream>
using namespace std;
class GInt {
    string _tab;
public:
    GInt(int entier = 0);
    GInt(string entier = "0");
    GInt(const GInt& other);  // Constructeur de copie.
    ~GInt();
    friend ostream& operator<<(ostream& os)
    GInt operator+(const GInt op1, const GInt op2);
    string getValue() const;
};
```

+2

NOM Poupa

Prénom

Promo

Date

3/3

MATIÈRE

Fichier gint.cpp

```
#include <iostream>
#include "gint.h"
using namespace std;

GInt::GInt(int entier) {
  string entierCaste = (string) entier;
  // On passe de 576 à "576" par exemple,
  //conversion entier => string
        *_tab = new string[entierCaste.length()];
  // Allocation dynamique
  *_tab = entierCaste;
}

GInt::GInt( string entier) {
  *_tab = new string[entier.length()];
  *_tab = entier;
}

GInt::GInt(const& GInt otherInt) {
  _tab = strdup(otherInt.__tab);
  // _tab dynamique, on fait une copie via strdup
}
```

+1

```
GInt::~GInt() {
delete _tab;
// _tab n'est jamais nul grâce aux valeurs par défaut
// du constructeur
}

ostream& operator<<(ostream& os) {
cout << _tab << endl;
return os;
}

~~if(i_tab != NULL){~~
```

```
GInt GInt::operator+(const GInt op1, const GInt
                                        op2) {
  GInt resultat;

  return resultat;
}
```

```
string GInt::getValue() const {
  return _tab;
}
```