ÉCOLE POLYTECH'NICE SOPHIA
SI4
2021-2022

Julien Deantoni

Programmation Multi Paradigmes : CPP

TD numéro 2

Gestion d'objets alloués statiquement en mémoire et création de classes simples

Objectif

Dans un premier temps l'objectif de cet exercice est de manipuler les différentes manières de passer des paramètres et de comprendre les implications de ces choix. Dans un deuxième temps, l'idée est de réaliser une classe simple. Cette dernière permettra de comprendre l'implication des choix des types (objets, pointeurs / références) sur la conception.

Énoncé des problèmes

À réfléchir sans l'ordi...

Voici un ensemble de programmes simples. À vous de donner le résultats des exécutions suivantes sur la sortie standard. Vous montrerez aussi ce qu'il se passe en mémoire.

exo1

```
...
using namespace std;

int main(){

  string s = "toto";
  cout << s << endl;

  string* pt_s;
  cout << (*pt_s) << endl;
  cout << pt_s << endl;
  cout << &pt_s << endl;

  pt_s = &s;
  cout << pt_s << endl;
  cout << &pt_s << endl;

 return EXIT_SUCCESS;
}
```

Maintenant que vous avez compris ce bout de programme, quelle est votre critique ? pourquoi ne faut-il jamais faire cela ? Quelle pourrait être une ligne dangereuse ?

exo2

```
...
using namespace std;

int do_something(string s){
  int i = 06;
  s = s + '!';
  return i;
}


int main(){

  string s = "yeah ";
  cout << s << endl;

  do_something(s);
  cout << s << endl;
  cout << i << endl;

  int j = 29;
  j = do_something(s);
  cout << s << endl;
  cout << j << endl;


 return EXIT_SUCCESS;
}
```

Maintenant que vous avez compris ce bout de programme, quelle est votre critique ? Y a t'il une ligne dangereuse ?

exo3

```
...
using namespace std;

string* do_something(string s){
  string* pt_s = &s;
  s = s + '!';
  return pt_s;
}


int main(){

  string s = "yeah ";
```

```
  cout << s << endl;

  do_something(s);
  cout << s << endl;

  string* pt_s = do_something(s);
  cout << pt_s << endl;
  cout << (*pt_s) << endl;


 return EXIT_SUCCESS;
}
```

Comme avant : critiques ? problèmes potentiels ? solution ?

exo4

```
...
using namespace std;

string* do_something(string* s){
  string* pt_s = s;
  (*s) = (*s) + '!';
  return pt_s;
}


int main(){

  string s = "yeah ";
  cout << s << endl;

  do_something(&s);
  cout << s << endl;

  string* pt_s = do_something(&s);
  cout << (*pt_s) << endl;


 return EXIT_SUCCESS;
}
```

Comme avant : critiques ? problèmes potentiels ? solution ?

exo5

```
...
using namespace std;
```

```
string* do_something(string& s){
  string* pt_s = &s;
  s = s + '!';
  return pt_s;
}


int main(){

  string s = "yeah ";
  cout << s << endl;

  do_something(s);
  cout << s << endl;

  string* pt_s = do_something(s);
  cout << (*pt_s) << endl;


 return EXIT_SUCCESS;
}
```

Et maintenant ?

exo6

```
...
using namespace std;

string do_something(string& s){
  s = s + '!';
  return s;
}


int main(){

  string s = "yeah ";
  cout << s << endl;

  do_something(s);
  cout << s << endl;

  string s2 = do_something(s);
  cout << s2 << endl;

  cout << &s  == &s2 << endl;

 return EXIT_SUCCESS;
}
```

alors ?

exo7

```
...
using namespace std;

string& do_something(string s){
  s = s + '!';
  return s;
}


int main(){

  string s = "yeah ";
  cout << s << endl;

  do_something(s);
  cout << s << endl;

  string s2 = do_something(s);
  cout << s2 << endl;

  cout << &s == &s2 << endl;


 return EXIT_SUCCESS;
}
```

et cette fois ?

exo8

```
...
using namespace std;

string do_something(string** s){
  string* s2 = *s;
  **s = *s2 + '!';
  return **s;
}


int main(){

  string s = "yeah ";
  cout << s << endl;

  do_something(&(&s));
```

```
    cout << s << endl;

    string* s0 = &s;
    string s2 = do_something(&s0);
    cout << s2 << endl;

    cout << &s  << endl;
    cout << &s2 << endl;


   return EXIT_SUCCESS;
  }
```

finalement ?

Un classe et son main...

attention, implémentation à en 3 étapes

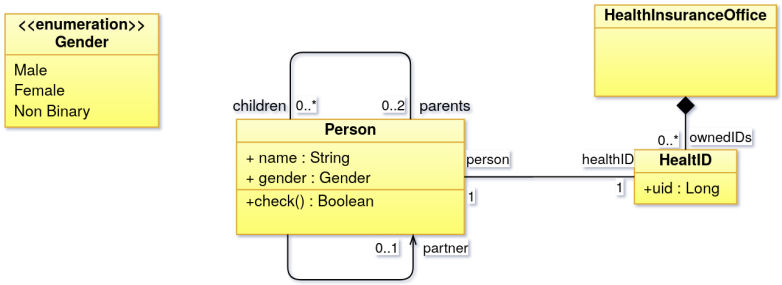

Figure 1: Représentation partielle de l'énoncé

On désire implémenter une classe `Personne`. Une personne possède un genre et un prénom. Le genre peut être "Masculin" ou "Féminin".
Implémentez dans un premier temps cette classe simple et créez dans le `main` deux personnes nommées Charles et Amandine et respectivement de genre Masculin et Féminin. Lors de la définition de cette classe posez vous la question de quels sont les constructeurs qui sont nécessaires. Vous vous poserez également la question de savoir quel est le type exact de chacun des attributs de la classe Personne en prenant soin de savoir justifier votre choix.
Afin de faciliter la visualisation de votre programme, vous allez surcharger l'opérateur d'affichage de la class *Person*. Pour ce faire, créez une fonction non membre de la classe `Person` dans *Person.h* dont la déclaration est la suivante:

```
std::ostream& operator<<(std::ostream& os, const Person& p);
```

vous pourrez, une fois cette fonction implémentée afficher une personne de la manière suivante:

```
Person p;
std::cout << p;
```

cette fonction non membre de la classe `Personne` représentera le plus précisément possible une Personne. Une Personne peut avoir zéro ou une Personne pour partenaire (si vraiment vous le désirez, vous pourrez dans une version ultérieur supposer qu'une Personne possède de 0 à n partenaire... Pour l'instant on se limitera à 0 ou 1). De plus, une Personne peut connaître ou non ses parents. Elle n'a jamais plus de deux parents. Une Personne peut également avoir des enfants (0 ou plus).
Réalisez la Classe Personne. Ensuite, arrangez-vous pour reflèter ces différentes étapes:

- Charles et Amandine sont partenaires l'un de l'autre. De leur union né un enfant : Fred.
- Charles et Amandine se séparent. Charles retrouve une partenaire : Sylvie. Ils font un enfant : Régis
- Amandine retrouve un partenaire : Jeff. Ils font deux enfants : Marie et Pedro.
- Marie et Régis finissent par se rencontrer et deviennent partenaire. Ils ont un enfant : Benoit.

Une fois les classes et tous leurs constructeurs corrects, Implémenter une fonction membre de la classe Personne qui vérifie un maximum de contraintes sur les objects de cette classe (pas + de deux parents, pas de cycles entre parents et enfants, pas d'enfants incestueux, etc). Cette fonction (nomée `check()`) renverra un booléen et affichera ou non un ou plusieurs messages d'erreurs sur la sortie standard en fonction d'un paramètre booléen.
Dans une deuxième version, il existe un classe simple représentant la caisse primaire d'assurance maladie. Elle contient un ensemble d'identifiants uniques qui eux même sont liés à une personne dès leur création. Modifiez votre programme pour refléter ce changement.
Dans une troisième version, une personne a aussi connaissance de son identifiant unique auprès de la caisse d'assurance maladie. Celui-ci est attribué à toute personne dès la naissance.