Trimestre Automne, 2018

Mohamed Lokbani

IFT1169 – Examen Final –

Inscrivez tout de suite : votre nom et le code permanent.

Nom : _____| Prénom(s) : _____|

Signature : _____| Login : _____|

Date : Vendredi 14 décembre 2018

Durée : 3 heures (de 16h30 à 19h30)

Local : Z-330, Pavillon Claire McNicoll

Directives :

- Toute documentation est permise.
- Ordinateur, Téléphones **non** permis.

- Répondre directement sur le questionnaire.
- Les réponses **doivent être brèves, précises, claires et nettement présentées.**

1._____/12

2._____/18

3._____/10

4._____/15

5._____/20

6._____/25

Total : _____/100

Directives officielles

* Interdiction de toute communication verbale pendant l'examen.

* Interdiction de quitter la salle pendant la première heure.

* L'étudiant qui doit s'absenter après la première heure remettra sa carte d'étudiant au surveillant, l'absence ne devant pas dépasser 5 minutes. Un seul étudiant à la fois peut quitter la salle.

* Toute infraction relative à une fraude, un plagiat ou un copiage est signalée par le surveillant au directeur de département ou au professeur qui suspend l'évaluation.

F.A.S

Exercice 1 (12 points) soit le fragment de code suivant :

```
01 class Alpha{
02 public:
03         Alpha(int i) {a_data = i;}
04 private:
05         int a_data;
06 };
07
08 class Bravo{
09 public:
10         Bravo(int i) {b_data = i;}
11 private:
12         Alpha b_data;
13 };
```

Sa compilation provoque cette cascade d'erreurs!

```
>g++ -Wall -pedantic -Os -std=c++17 -c exo0.cpp -o exo0.o

exo0.cpp: In constructor 'Bravo::Bravo(int)':
exo0.cpp:10:15: error: no matching function for call to 'Alpha::Alpha()'
  Bravo(int i) {b_data = i;}
               ^
exo0.cpp:3:2: note: candidate: Alpha::Alpha(int)
  Alpha(int i) {a_data = i;}
  ^
exo0.cpp:3:2: note: candidate expects 1 argument, 0 provided
exo0.cpp:1:7: note: candidate: constexpr Alpha::Alpha(const Alpha&)
 class Alpha{
       ^
exo0.cpp:1:7: note: candidate expects 1 argument, 0 provided
exo0.cpp:1:7: note: candidate: constexpr Alpha::Alpha(Alpha&&)
exo0.cpp:1:7: note: candidate expects 1 argument, 0 provided
>Exit code: 1
```

Comme il est mentionné par les messages d'erreur, les « #include » ne sont pas la cause.

Expliquez ces erreurs et corrigez le fragment de code en conséquence.

Exercice 2 (18 points) ce programme compile et s'exécute correctement.

```
01 #include <iostream>
02
03 class Alpha{
04 public:
05  virtual void affiche() {std::cout << "Alpha" << std::endl;}
06 };
07
08 class Bravo : public Alpha {
09 public:
10  virtual void affiche() {std::cout << "Bravo" << std::endl;}
11 };
12
13 class Charlie : public Bravo {
14 public:
15  virtual void affiche() {std::cout << "Charlie" << std::endl;}
16 };
17
18 void affiche1(Alpha* f){
19  f->affiche();
20 }
21
22 void affiche2(Alpha& f){
23  f.affiche();
24 }
25
26 void affiche3(Alpha f){
27  f.affiche();
28 }
29
30 int main() {
31
32  Alpha* a =  new Bravo;
33  Bravo& b = *new Charlie;
34  Bravo  c = *new Charlie;
35
36  affiche1(a);
37  affiche1(&b);
38  affiche1(&c);
39
40  std::cout << std::endl;
41
42  affiche2(*a);
43  affiche2(b);
44  affiche2(c);
45
46  std::cout << std::endl;
47
48  affiche3(*a);
49  affiche3(b);
50  affiche3(c);
51
52  std::cout << std::endl;
53
54  return 0;
55 }
```

Que va afficher en sortie le précédent programme? Expliquez brièvement votre démarche.

Exercice 3 (10 points) le fragment de code suivant est syntaxiquement correct :

```
01      int x = 4;
02      auto y = [&r = x, x = x+1]()->int {
03              r += 2;
04              return x+2;
05           }();
06
07      std::cout << "x: " << x << " ; y: " << y << std::endl;
```

Tout en expliquant votre démarche, quel sera l'affichage obtenu en sortie à la ligne « 07 ».

Exercice 4 (15 points) ce programme compile et s'exécute correctement.

```
01 #include <iostream>
02 #include <algorithm>
03 #include <vector>
04 #include <iterator>
05
06 class Tester {
07 double valeur;
08 public:
09         Tester(double epsilon):valeur(epsilon) {}
10         double operator()(double val)  {
11                     if (val < valeur) {
12                                 return 0;
13                         } else {
14                                 return val;
15                         }
16         }
17 };
18 void test(std::vector<double>& v, const double& epsilon) {
19     std::transform(v.begin(), v.end(), v.begin(),Tester(epsilon));
20 }
21 int main() {
22     std::vector<double> alpha = {1.1, 2.2, 3.3, 4.4, 5.5};
23     test(alpha, 2.9);
24     copy(alpha.begin(),alpha.end(),std::ostream_iterator<double>(std::cout," "));
25     std::cout << std::endl;
26     return 0;
27 }
```

Dans cet exemple, l'algorithme « transform » applique une transformation (le 4e argument représenté par le functor) à chaque élément du vecteur « alpha » (les deux premiers arguments) et les insère dans le même vecteur à partir du début (3e argument). L'affichage obtenu en sortie est : « `0 0 3.3 4.4 5.5` ». Ainsi, toutes les valeurs inférieures à « `2.9` » sont remplacées par la valeur « `0` ».

On vous demande de réécrire le 4e argument de « transform » en utilisant à la place du « functor » une fonction Lambda.

Exercice 5 (*20 points*) soit le programme suivant :

```
01 #include <iostream>
02
03 class Un_Ptr {
04 public:
05     Un_Ptr(int* p) : courant(p) {}
06     int operator*() const { return *courant; }
07 private:
08     int* courant;
09 };
10
11 int main() {
12     int tab[] = {1, 2, 3};
13     Un_Ptr ptr = tab;
14     std::cout << *ptr << std::endl; // 1
15
16     *ptr = 10;
17     std::cout << *ptr << std::endl; // 10
18
19     return 0;
20 }
```

5.1 Le programme va afficher ce message d'erreur à la compilation.

```
>g++ -Wall -pedantic -Os -std=c++17 -c exo49.cpp -o exo49.o
exo49.cpp: In function 'int main()':
exo49.cpp:16:10: error: lvalue required as left operand of assignment
     *ptr = 10;
          ^
>Exit code: 1
```

Comme il est mentionné dans le message d'erreur, l'affectation de la ligne « 16 » pose problème. Expliquez les raisons et modifier le contenu de la classe « **Un_ptr** » pour permettre l'instruction de la ligne « 16 » du programme.

5.2 Après avoir apporté les corrections nécessaires, on veut généraliser l'utilisation de la classe « Un_Ptr » pour traiter un type quelconque. Modifiez la classe « Un_Ptr » afin de traiter, par exemple, la fonction « main » suivante :

```
01 int main() {
02     int tab[] = {1, 2, 3};
03     Un_Ptr<int> ptr = tab;
04     std::cout << *ptr << std::endl; // 1
05
06     *ptr = 10;
07     std::cout << *ptr << std::endl; // 10
08
09     return 0;
10 }
```

5.3 Complétez maintenant la classe générique « Un_Ptr<T> » avec les méthodes nécessaires pour traiter la fonction « main » suivante :

```
01 int main() {
02     int tab[] = {1, 2, 3};
03     for (Un_Ptr<int> p = tab; p != tab + 3; ++p) {
04         std::cout << *p << " ";  // 1 2 3
05     }
06     std::cout << std::endl;
07     std::string deschaines[] = {"Ceci", "est", "un", "exemple", "en", "C++"};
08
09     for (Un_Ptr<std::string> p = deschaines; p != deschaines + 6; ++p) {
10         std::cout << *p << " "; // Ceci est un exemple en C++
11     }
12
13     std::cout << std::endl;
14     return 0;
15 }
```

Exercice 6 (25 points) dans cet exercice, vous allez examiner les différentes techniques permettant de copier les valeurs d'une « map » vers un vecteur.

Soit la « map » :

```
std::map<std::string, int> motMap = { { "image", 6 }, { "tournage", 5 },
       { "hauteur", 9 }, { "action", 6 }, { "origine", 2 }, { "heure", 1 } };
```

On déclare le vecteur d'entiers, « vecDeValeurs », ayant une taille fixe, le nombre d'éléments dans la « map » :

```
std::vector<int> vecDeValeurs;
vecDeValeurs.reserve(motMap.size());
```

À la fin de l'opération de copie, le vecteur « vecDeValeurs » va contenir {6 , 9 , 1 , 6 , 2 , 5}, les valeurs de la « map ».

6.1 Écrire le frgament de code qui permet de copier les valeurs de « motMap » dans « vecDeValeurs » en utilisant une technique combinant « for » et « auto ».

6.2 Écrire le frgament de code qui permet de copier les valeurs de « motMap » dans « vecDeValeurs » en combinant l'algorithme « for_each » avec une fonction Lambda.

6.3 Écrire le frgament de code qui permet de copier les valeurs de « motMap » dans « vecDeValeurs » en combinant l'algorithme « transform » et une fonction Lambda. Nous avons expliqué le fonctionnement de « transform » dans l'exercice 4.

6.4 Écrire le frgament de code qui permet de copier les valeurs de « motMap » dans « vecDeValeurs » en combinant l'algorithme « transform » avec un pointeur de fonction (pas de functor). Vous devez définir par la même occasion cette fonction. L'idée est de sortir le code de la fonction Lambda de la précédente question (« 6.3 ») à l'extérieur et de voir comment passer la fonction à « transform ».

Joyeuses Fêtes et bonne année!