Trimestre Hiver, 2008

Mohamed Lokbani

IFT1169 – Examen Intra –

Inscrivez tout de suite votre nom et le code permanent.

Nom: _____| Prénom(s): _____|

Signature: _____| Code perm: _____|

Date : mardi 26 février 2008

Durée : 2 heures (de 16h30 à 18h30)

Local : Z-209 ; Pavillon Claire McNicoll

Directives:

- Toute documentation permise.
- Calculatrice **non** permise.

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

1._____/20 (1.1, 1.2, 1.3, 1.4, 1.5)

2._____/20 (2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9)

3._____/15 (3.1, 3.2)

4._____/20 (4.1, a, b, c, d, 4.2)

5._____/25 (5.1)

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 (20 points)

1.1 Quelles sont les **trois** façons de faire des commentaires en C++ et en quoi peuvent-elles être différentes l'une de l'autre?

1.2 Définir ce qu'est « une copie superficielle » et donner un court exemple.

1.3 Pourquoi faut-il préférer le compilateur au préprocesseur ?

1.4 Comment peut-on empêcher un programmeur donné d'utiliser le constructeur de recopie ainsi que l'opérateur d'affectation lorsqu'on définit une classe donnée?

1.5 Le pointeur « this » fait-il référence à quelle relation dans les instructions suivantes :

```
class Nombre{
private:
    int valeur;
public:
    void SetNombre(int);
    Nombre* getNombre();

};
void Nombre::SetNombre(int valeur){
    this->valeur = valeur;
}
Nombre* Nombre::getNombre(){
    return this;
}
```

Exercice 2 (20 points) Expliquer brièvement les déclarations suivantes :

2.1 `int* x ;`

2.2 `int* x[10] ;`

2.3 `int *(x[10]);`

2.4 `int **x;`

2.5 `int (*x)[10];`

2.6 `int *fonction() ;`

2.7 `int (*fonction)() ;`

2.8 `int (*(*fonction())[10])() ;`

2.9 `int (*(*x[4])())[5] ;`

Exercice 3 (15 points)

3.1 Tout en justifiant votre réponse, est-ce que le programme suivant compile-t-il correctement ?

```
01
02          #include <iostream>
03          using namespace std;
04
05          namespace {
06              int p = 1;
07          }
08
09          void fonction(){
10              ++p;
11          }
12
13          namespace Un{
14              namespace{
15                  int p;
16                  int q = 3;
17              }
18          }
19
20          using namespace Un;
21
22          void test(){
23              ++p;
24              Un::++p;
25              cout<<"++q = "<< ++q << endl;
26          }
27
28          int main(){
29              test();
30              return 0;
31          }
```

(a) oui, il compile correctement | (b) non, il y a un problème à la compilation |

Pourquoi ?

3.2 Tout en justifiant votre réponse, est-ce que le programme suivant compile-t-il correctement ?

```
01          include <iostream>
02          using namespace std;
03
04          int main(){
05
06              namespace local {
07                  int k;
08              }
09
10              return 0;
11          }
```

(a) oui, il compile correctement | (b) non, il y a un problème à la compilation |

Pourquoi ?

Exercice 4 (20 points) Soit le programme suivant qui compile et s'exécute correctement :

```
01  #include <iostream>
02
03  using namespace std;
04
05  template <class T>
06  T Test(T Var1) {
07      cout << "-1-" << endl;
08      return Var1;
09  }
10
11  template <class T>
12  T* Test(T *Var2) {
13      cout << "-2-" << endl;
14      return Var2;
15  }
16  const char* Test(const char *Var3) {
17      cout << "-3-" << endl;
18      return Var3;
19  }
20
21  int main() {
22
23      int p = 5;
24      int q = Test(p);
25      double r = Test(3.1234);
26
27      cout << "q = " << q << endl;
28      cout << "r = " << r << endl;
29
30      int *s = Test(&p);
31      char *t = "Un!";
32
33      cout<<"s = " << *s << endl;
34      cout<<"t = "<< t << endl;
35
36      const char *u = Test("Deux!");
37
38      cout << "u = " << u << endl;
39
40      return 0;
41  }
42
```

4.1 Tout en justifiant votre réponse, citer la signature de la fonction « Test » générique ou classique, appelée par les lignes suivantes :

(4.1.a)	ligne 24	int q = Test(p);

Nom de la fonction ?

(a)	T Test(T)	(b)	T* Test(T*)	(c)	const char* Test(const char*)

Pourquoi ?

(4.1.b) ligne 25 | `double r = Test(3.1234);`

Nom de la fonction ?

(a) | `T Test(T)` | (b) | `T* Test(T*)` | (c) | `const char* Test(const char*)`

Pourquoi ?

(4.1.c) ligne 30 | `int *s = Test(&p);`

Nom de la fonction ?

(a) | `T Test(T)` | (b) | `T* Test(T*)` | (c) | `const char* Test(const char*)`

Pourquoi ?

(4.1.d) ligne 36 | `const char *u = Test("Deux!");`

Nom de la fonction ?

(a) | `T Test(T)` | (b) | `T* Test(T*)` | (c) | `const char* Test(const char*)`

Pourquoi ?

4.2 Tout en justifiant votre réponse, que va afficher en sortie le précédent programme ?

Exercice 5 (25 points) Le but de cet exercice est de montrer comment lever une exception suite à une erreur de conversion de type. Pour cela, vous allez écrire un programme complet en C++ dans un seul fichier « **exo5.cpp** » en tenant compte de la description suivante :

Soit la classe « **Forme** » qui ne contient qu'une seule méthode : la méthode virtuelle « **fonction** ». La méthode « **fonction** » est une méthode constante qui n'accepte aucun argument et qui ne retourne rien.

Soit la classe « **Triangle** » qui hérite de manière publique de la classe « **Forme** ». La classe « **Triangle** » ne contient qu'une seule méthode : la méthode virtuelle « **fonction** ». La méthode « **fonction** » est une méthode constante qui n'accepte aucun argument et qui ne retourne rien.

La méthode « **main** » contient les instructions suivantes:

-1- La variable « **UneForme** », une instance de « **Forme** ».

-2- La variable « **ref_Forme** », c'est une référence vers une instance de « **Forme** ». Elle sera initialisée avec l'instance « **UneForme** ».

-3- La variable « **ref_Triangle** », c'est une référence vers une instance de « **Triangle** ». Elle sera initialisée avec l'instance de « **ref_Forme** » après sa conversion en utilisant un des opérateurs appropriés pour ce genre d'opération. Cette déclaration peut lever une exception, il est donc préférable de bien la cadrer par les mesures appropriées.

-4- Si une exception est levée par la précédente déclaration, nous devons prévoir les instructions nécessaires pour la capturer.
La classe « exception » sert comme base pour toutes les exceptions levées par certaines instructions d'un programme donné. Ainsi, parmi les exceptions levées directement par la classe « exception » nous avons :

bad_alloc	Lancée par l'opérateur « new » s'il se produit une erreur lors de l'allocation.
bad_cast	Lancée par « dynamic_cast » s'il se produit une erreur lors de la conversion.
bad_exception	Lancée si l'exception n'est pas capturée par aucun « catch ».
bad_typeid	Lancée par « typeid » quand l'opérande associée à « typeid » est un pointeur NULL.

Écrire le « catch » approprié à l'instruction décrite dans -3-. Ce « catch » doit afficher en sortie un message informatif sur la raison de l'erreur.

-5- Après avoir écrit votre programme, expliquer brièvement votre démarche et les choix retenus pour réaliser cet exercice.