Université de Reims Champagne-Ardenne
UFR Sciences Exactes et Naturelles
Licence Informatique
INFO0402: Programmation Orientée Objet

TD N°1

Exercice 1: Modèle de données

1. A quoi correspond le modèle de données LP64 ?
2. Si je compile un programme écrit sur un système avec un modèle de données LP64, et que je le recompile sur un système avec un modèle de données ILP32, quels sont les changements que je peux observer et quels problèmes cela peut-il poser ?
3. pourquoi est-il utile d'avoir des types entiers à longueur fixe ?

Exercice 2: Conversions

1. Qu'est ce qu'une promotion numérique ?
2. Une promotion numérique est-elle toujours sans perte de précision ?
3. Que se passe-t-il si l'on convertit une entier non signé vers entier non signé plus petit ?
4. Soit le code suivant :

```
unsigned char c1 = 100, c2 = 3, c3 = 4;
unsigned char r1 = c1*c2;
unsigned char r2 = r1 / c3;
unsigned char r = c1 * c2 / c3;
```

   Donner la valeur de `r1`, `r2` et `r`.
5. Existe-t-il des entiers qui peuvent être convertit en flottant exactement ?
6. Existe-t-il des flottants qui peuvent être convertit en entier exactement ?
7. Quelle est la différence entre la troncature d'un flottant en entier, et les fonctions d'arrondi `floorf` et `ceilf` ?

Exercice 3: Comprendre `lvalue` et `rvalue`

Rappel : dans ce cours, on utilise les sens de `lvalue` et `rvalue` donnés à partir du C_{11}^{++}, et qui ne recouvrent pas le sens qu'on leurs donne habituellement.

1. Donner le sens de `lvalue` et `rvalue`.
2. Quelle est la différence entre une variable sans modificateur et avec les modificateurs `&`, `&&` et `*` ?
3. Pourquoi, lorsque je déclare une variable de type référence sur une `rvalue`, alors cette variable est une `lvalue` ?

```
int a = 5;
int &b = a;
int c = b;
int d = (a+4)/2;
int &e = 5;
int &f = a/2;
int &&g = a;
int &&h = b;
int &&i = a/4;
int &&j = 8;
```

code 1

```
int    fun1(), &fun2(), &&fun3();
int A = fun1();
int &B = fun2(a);
int &&C = fun3(5);
int D = fun2(a);
int E = fun3(7);
int &F = fun3(a);
int &G = fun1(a);
int &&H = fun2(a);
int &&I = fun1();
```

code 2

4. Dans le code 1 ci-dessus, pour chaque ligne, identifier si dans les parties gauche et droite des définition, ce sont des `lvalue`s ou des `rvalue`s, puis en déduire si c'est une expression C_{11}^{++}valide.
5. Dans le code 2 ci-dessus, pour chaque ligne, identifier si dans les parties gauche et droite des définitions, ce sont des `lvalue`s ou des `rvalue`s, puis en déduire si c'est une expression C_{11}^{++}valide.
6. Quel est le sens du qualificateur `const` lorsqu'il est utilisé dans la définition d'une variable de type `int`, `int&`, `int&&` ?
7. Que deviennent les affectations de la question 3 si l'on qualifie `const` toutes les variables ?

Exercice 4: Unité de traduction, durée de stockage et liens

1. Rappeler ce qu'est une unité de traduction.
2. Donner un exemple d'un code contenant des objets ayant des durées de stockage automatique, statique et dynamique. On essayera de donner des plusieurs exemples pour chaque durée de stockage.
3. Donner un exemple de code modulaire contenant des objets sans lien, un lien interne, un lien externe.
4. Soit le code suivant :

```
void fun() {
    char *a = "tralala", b[] = "tralala";
    a[0] = 'T';
    b[0] = 'T';
}
```

   Pourquoi ce code provoque-t-il une erreur de segmentation ?

Exercice 5: Surcharge de fonction 1

1. Est-il possible en C++ de faire en sorte que la fonction `max` fonctionne à la fois pour des entiers et des flottants sans utiliser ni les macros du préprocesseurs ni les templates.
2. Pourquoi l'appel `max(3,4.5f)` échoue-t-il alors ?
3. Pourquoi a-t-on intérêt à définir cette fonction inline ?
4. On veut écrire une seule fonction `Rand` qui permet les appels suivants :
   - `Rand()` retourne un nombre aléatoire entre 0 et 1.
   - `Rand(6)` retourne un nombre aléatoire entre 0 et 6.
   - `Rand(2,10)` retourne un nombre aléatoire entre 2 et 10.

   Expliquer comment écrire une telle fonction.
5. Pourquoi serait-il préférable que cette fonction soit `inline` ?

Exercice 6: Surcharge de fonction sur un type qualifié et modifié

La table et les règles qui permettent de répondre à ces questions sont à la page 29. Pour les combinaisons de surcharges suivantes, on se pose les deux questions :

a ces surcharges sont-elles possibles ?
b quels sont les types captés par les différentes surcharges, et quelles sont ceux qui ne le sont pas ?

1. `void fun(int)` et `void fun(const int)`
2. `void fun(int)` et `void fun(int&)`
3. `void fun(int)` et `void fun(int&&)`
4. `void fun(const int)` et `void fun(int&)`
5. `void fun(int&)` et `void fun(const int&)`
6. `void fun(int&)` et `void fun(int&&)`
7. `void fun(int&)` et `void fun(const int&&)`
8. `void fun(const int&)` et `void fun(int&&)`
9. `void fun(const int&)` et `void fun(const int&&)`
10. `void fun(int&&)` et `void fun(const int&&)`

Exercice 7: Macro du précompilateur

1. soit le macro suivante : `#define MYMACRO(a,b) if (a) fun(b)`. Pourquoi cette macro peut-elle poser des problèmes et comment le corriger ?
2. soit le macro suivante : `#define abs(x) ((x)>=0 ? (x) : -(x))`. Pourquoi cette macro peut-elle poser des problèmes et comment le corriger ?
3. soit le macro suivante :

```
#define MYMACRO(a,b) \
    instruction1; \
    instruction2; \
    /*...*/ \
    instructionN;
```

   Pourquoi cette macro peut-elle poser des problèmes et comment le corriger ?