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

TD N°1

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 ?

Solution:

1. cf section 4.1 du chapitre 1.
2. construire un exemple avec :
   durée de stockage automatique : variable locale, variable dans un sous-bloc, paramètre d'une fonction
   durée de stockage statique : variable globale, variable statique
   durée de stockage dynamique : dynamique : allocation dynamique
3. construire un exemple avec :
   objet sans lien : variable locale dans une fonction.
   objet avec lien interne : appel d'une fonction ou d'une variable globale définie dans l'unité de traduction.
   objet avec lien externe : appel d'une fonction ou d'une variable globale définie dans une autre unité de traduction.
4. `a` pointe vers une zone de mémoire constante (zone `text` du programme), et sa modification provoque l'erreur.
   `b` est un tableau dans le stack (comme une variable locale) initialisé avec la chaine "tralala". Donc, pas de problème.

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` ?

Solution:

1. en utilisant les surcharges de fonction, int max(int,int) et float max(float,float).
2. le compilateur n'est pas en mesure de déterminer quelle fonction appeler.
3. car c'est une fonction courte, et qu'un appel de fonction est un gaspillage de ressources dans ce cas.

4. `int Rand(int u=0,int v=0)`
   si $u = v$, alors la loi est sur $[0, 1]$, si $u > v$ alors la loi est sur $[0, u]$, si $u < v$ alors la loi est sur $[u, v]$.
   Afin que cela fonctionne sur tous les entiers, remplacer 0 par `std::numeric_limits<int>::min()`.
5. Petite fonction possiblement très fréquemment appelée.

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 ?

Solution:

1. si on écrit :
```
if (x) MYMACRO(3,4)
else x++;
```
   alors le code compris par le compilateur est :
```
if (x) {
    if (a) fun(b) else x++;
}
```
2. `MYMACRO(fun(x))`
   alors le code compilé est :
   `((fun(x))>=0 ? (fun(x)) : -(fun(x)))`
   Donc, `fun(x)` est appelée deux fois.
   Utiliser une fonction inline qui fait la même chose.
3. La solution n'est pas simple :
   — Avec `while (x) MYMACRO(a,b);`, seule la première instruction est exécutée par le `while`.
   — si ajoute des accolades autour des instructions, alors l'écriture :
   `if (whatever) MYMACRO(foo, bar); else baz;`
   provoque une erreur de compilateur (en raison du ; après le bloc).
   — une solution serait de placer les instructions dans le bloc :
```
#define MYMACRO(a,b) \
    do { \
      instruction1; \
      instruction2; \
      /*...*/ \
      instructionN; \
    } while(false)      // pas de ; ici
```
   alors l'écriture `if (whatever) MYMACRO(foo, bar); else baz;` provoque le résultat attendu.

   Conclusion :

   — dans la plupart des cas, on peut trouver un appel de la macro qui a un comportement non souhaitable.
   — éviter d'utiliser des macros sauf lorsque cela est vraiment nécessaire, et ne peuvent pas être réalisée en utilisant des fonctions inline ou des templates C++.
   — le seul cas où l'on ne peut vraiment pas s'en passer est pour la réalisation de code dans la compilation conditionnelle.