Analyse et Programmation Orientées Objets / C++

Extensions du langage C

Le modificateur *const* (1)

➡ **Besoins et problèmes posés**

- Améliorer l'usage de la directive **#define**

- Déclarer des constantes avec un sens fonctionnel fort
- Analogie recherchée avec la déclaration des variables
- Contrôle du type de la constante
- Limiter la portée de la déclaration
- Assurer les transferts de la constante par VALEUR

Le modificateur *const* (2)

➡ **Règles d'utilisation**

- Portée limitée au bloc qui définit l'invariant

- Emploi du mot clé ***const*** à gauche de la déclaration
- Typage vérifié à la compilateur (pas pour les littéraux)

Le modificateur *const* (3)

- **Exemple de déclaration de constantes**

```
const float fPI=3.14159;
char* const pCH1   = "AAAA";
const char pCH2 [] = "BBBB";
```

Le modificateur **const** doit être placé tout à gauche de ce qui ne doit pas être modifié.

Modificateur *const* (4)

- **Usage dans la signature d'une fonction**

Fonctions *inline* (1)

- **Besoins et problèmes posés**
  - Améliorer l'usage de la directive ***#define***
  - Conserver la possibilité de macro-définitions
  - Analogie recherchée avec l'usage des fonctions
  - Eviter de mettre en place le mécanisme d'appel
  - Contrôler la signature effective / signature formelle

Fonctions *inline* (2)

➡ **Règles d'utilisation**

- Bon usage limité à un corps de 2-3 lignes instructions
- Emploi du mot clé ***inline*** à gauche de la définition
- Typage vérifié par le compilateur (pas pour les macros)
- Ligne d'appel substituée par le code binaire généré

Contrainte : définition uniquement (header)

Fonctions *inline* (3)

- **Exemple de définition d'une fonction**

```
inline float surface (float rayon) {
        return rayon*rayon*PI;
}
```

Le type référence (1)

➡ **Besoins et problèmes posés**

- Simplifier la syntaxe du passage par adresse
- Aucune plus value sémantique
- Possibilité de créer des alias de paramètres
- Limiter l'emploi de la notation *
- Conserver tous les contrôles des lignes d'appel
- Ne pas altérer la lisibilité des codes sources

Le type référence (2)

➡ **Règles d'utilisation**

- Usage limité pour l'essentiel au passage par adresse
- Emploi du meta-caractère & dans la déclaration de type
- Usage dans les signatures formelles
- Le caractère implicite spécifié que dans le prototype
- Portée étendue implicitement à droite
- Typage vérifié par le compilateur

Le type référence (3)

- **Exemple de déclaration par référence**

```
int k=0;
int& k1=k;          // k1 est l'alias de k
int* pK;

k1=22;              // Modifie donc le contenu de k !
pK=&k1;             // pK reçoit l'adresse de k
```

Le type référence (4)

- **Paramètre formel de type référence**

```
void modifier (int& nX) { nX = 22; }

void main () {
int k=11;
      modifier (k);
      printf ("%d\n", k);
}
```