Compléments de C
Qualificateur const

Nicolas Gazères

Développeur Dassault Systèmes
ngs@3ds.com

Objet *const*

- Principe
  - (a) un objet déclaré *const* est *non-modifiable* une fois initialisé.
  - (b) il doit être initialisé dès sa déclaration.

```
int i ;                 // OK avec (a), i est non-const
i++ ;                   // OK i non-const

const int j ;           // KO !! cf. (b)

const int k = 2 ;       // OK     cf. (b)
k++;                    // KO !! cf. (a)
```

- Intérêt
  - Dès qu'une variable est déclarée *const*, le compilateur:
    - traque toute instruction de modification de la variable
    - détecte une erreur si c'est le cas.

Pointeur sur objet *const*

- Lorsqu'on définit un pointeur sur *objet const* :
  - C'est l'objet pointé qui est *const*.
  - Le pointeur, lui, est tout-à-fait modifiable.

```
int i;
const int *p ;              // (1) OK
const int *pi = &i;         // (2) OK
i++;                        // (3) OK i  est non-const
*pi++;                      // (4) OK pi est non-const
(*pi)++;                    // (5) KO *pi est const !

const int j = 3;
int * pci = &j;             // (6) KO conversion const→non-const
```

- Explications
  - (1) C'est compatible avec (b) car ce n'est pas le pointeur qui est *const*, c'est l'entier pointé. On n'est donc pas tenu d'initialiser le *pointeur*.
  - (2) On peut très bien affecter à un pointeur sur objet *const* l'adresse d'un entier *non-const*.
  - (4) Comme ++ est prioritaire sur *, c'est le pointeur qu'on modifie
  - (5) Là, c'est l'objet *const* pointé qu'on tente de modifier. cf. (a)
  - (6) On essaie de pointer vers un entier constant via un pointeur sur entier non-constant. C'est interdit car ça permettrait de contourner la protection sur l'entier.

Pointeur *const* sur objet

- Principe
  - (a) un objet déclaré *const* est *non-modifiable* une fois initialisé.
  - (b) il doit être initialisé dès sa déclaration
- Remarque
  - Si on définit un pointeur *const* sur objet
    - C'est le *pointeur* qui n'est plus modifiable.
    - l'objet pointé peut être *const* ou pas.
  - Syntaxiquement: le *const* doit être placé *après* l'étoile.

```
int i, j;
int * const cp ;                    // KO !! cf. (b)
int * const cpi = &i;               // OK (non-const → const)
(*cpi)++;                           // OK *cpi est non-const.
cpi = &j;                           // KO !! cpi est const. cf. (a)
*cpi++;                             // KO !! cpi est const. cf. (a)

const int * const cpci = &i;        // pointeur const sur entier const
cpci = &j;                          // KO !! cpci est un pointeur const
*cpci = 3;                          // KO !! *cpci est un const int.
```

#define ou *const* ?

- Les deux lignes suivantes ont en apparence le même effet:

```
#define MY_DOUBLE   3.7
const double MY_DOUBLE = 3.7;
```

- En fait, il y a des différences significatives:
  - *#define*
    - introduit une constante symbolique
    - qui est remplacée *au stade du préprocessing* partout où elle apparait dans le *code source* du programme
    - qui n'a donc pas d'impact sur la taille du programme
  - *const*
    - déclare comme non-modifiable une *variable*
      - en particulier, dont la valeur est *visible dans le débugger*

Recommandations

- Il faut utiliser *const* le plus souvent possible pour déclarer:
  - les *paramètres de fonction*
    - lorsqu'ils sont censés ne pas être modifiés par la fonction.
  - les valeurs de retour de fonction
    - lorsqu’ils sont censés ne pas être modifiés par l’appelant de la fonction.
  - les *constantes* du projet
    - lorsqu’on souhaite voir les valeurs dans le débugger.