

Exercice 1 : Connaissances générales

- (a) Quel est l'effet du mot-clé `static` sur une variable déclarée au **niveau fichier** ?
- A. Elle est allouée sur la pile
 - B. Elle est visible uniquement dans l'unité de traduction
 - C. Elle est réinitialisée à chaque appel de fonction
 - D. Elle doit être initialisée obligatoirement
- (b) Quel mot-clé doit être utilisé dans un fichier d'en-tête pour **déclarer** une variable globale définie dans une autre unité ?
- A. `static` B. `extern` C. `register` D. `volatile`
- (c) À propos de `volatile`, quelle affirmation est correcte ?
- A. Le compilateur peut mettre en cache la valeur dans un registre
 - B. Le compilateur doit relire la valeur en mémoire à chaque accès
 - C. `volatile` rend la variable constante
 - D. `volatile` supprime tout risque de data race
- (d) Considérez la déclaration `const int *p = {1,2,3};`. Quelle(s) affirmation(s) est/sont correcte(s) ?
- A. On peut modifier les éléments pointés par `p`
 - B. On ne peut pas modifier les éléments pointés par `p`
 - C. On peut modifier le pointeur `p` pour qu'il pointe vers un autre tableau
 - D. Les valeurs associées à `p` sont stockées dans une section de mémoire en lecture seule
 - E. Le code ne compile pas à cause de l'initialisation incorrecte de `p`
- (e) Une fonction arbore le prototype suivant `void f(const struct Data *const d);`. Quelles sont les implications de ce prototype ?
- A. La fonction peut modifier les champs de la structure pointée par `d` ;
 - B. La fonction peut modifier le pointeur `d` pour qu'il pointe vers une autre structure ;
 - C. La fonction ne peut pas modifier les champs de la structure pointée par `d` ;
 - D. La fonction utilise le passage par adresse par souci de performance ;
 - E. La fonction ne peut pas modifier l'adresse du pointeur `d` copié sur le stack.
- (f) Quel est l'effet de la déclaration `register int counter;` ?
- A. `counter` est stocké dans un registre et ne peut pas être adressé via un pointeur ;
 - B. `counter` est stocké dans un registre et peut être adressé via un pointeur ;
 - C. `counter` est une suggestion pour le compilateur de stocker la variable dans un registre, mais ce n'est pas garanti ;
 - D. `counter` est une variable globale accessible dans toutes les unités de traduction.
- (g) Comment accéder à une variable `int g` définie dans une autre unité de traduction ?
- A. `static int g;` C. `extern int g;`
B. `register int g;` D. `volatile int g;`
- (h) Laquelle de ces classes de stockage n'appartient pas au langage C ?
- A. `auto` B. `dynamic` C. `register` D. `static` E. `extern`
- (i) À quoi faire attention quand une structure est utilisée pour communiquer entre différentes architectures matérielles ?

- A. Boutisme/Endianess
- B. Alignement
- C. Padding
- D. Toutes les réponses ci-dessus

Exercice 2

Analysez le code suivant et indiquez ce qui est affiché sur la sortie standard.

```
#include <stdio.h>
int next_id(void) { static int id = 0; return ++id; }
int main(void) { printf("%d %d %d\n", next_id(), next_id(), next_id()); }
```

Exercice 3: Alignement et #pragma pack

- (a) Sur une machine 64-bit avec alignement naturel (char=1, short=2, int=4, double=8), calculez sizeof pour les structures ci-dessous.

```
typedef struct {
    char c;
    int i;
    short s;
} A;

#pragma pack(push, 1)
typedef struct {
    char c;
    int i;
    short s;
} B;

#pragma pack(pop)
```

- (b) Citez deux risques liés à l'utilisation de #pragma pack(1).

Exercice 4: Champs de bits

- (a) Déclarez une structure `Flags` contenant 3 indicateurs booléens: `ready`, `error`, `busy` (sur 1 bit), ainsi qu'un champ `code` sur 5 bits.
- (b) Écrire une fonction `set_error` qui reçoit un pointeur sur un `Flags` et met à 1 le flag `error` ainsi que met à 0 le flag `ready`:
- (c) Que se passe-t-il si on affecte la valeur 42 au champ `code` de la structure `Flags` ?