Identification /5

Prénom: .....

Nom: .....

Code permanent: .....

Veuillez choisir **une seule réponse** par question, avec une croix (×) ou un crochet (✓).

Q1. Une caractéristique souhaitable pour qu'une application informatique soit modulaire est que chaque composante règle une préoccupation précise. Quelle est cette caractéristique?

- ☐ Interchangeabilité
- ☐ Réutilisation
- ☐ Séparation
- ■ Spécificité

Q2. Quelle option du compilateur GCC permet de spécifier un symbole à la compilation?

- ■ `-D`
- ☐ `-S`
- ☐ `-d`
- ☐ `--symbol`

Q3. Qu'est-ce qui sera affiché sur la sortie standard par le programme suivant?

```
#include <stdio.h>

#define abs(X) ((X) >= 0 ? X : -(X))
#define dist(X, Y) abs(X - Y)

int main(void) {
  int a = 2;
  printf("%d\n", dist(-1, 5 - a));
  return 0;
}
```

- ☐ `-2`
- ☐ `4`
- ☐ `-6`
- ■ `8`

Q4. Considérons un module régulier C composé des fichiers `module.h` et `module.c`. Parmi les éléments suivants, lequel ne devrait pas apparaître dans le fichier `module.c`?

- ☐ La directive `#include "module.h"`
- ■ Une déclaration de fonction publique
- ☐ Une déclaration `typedef`
- ☐ Une macro-fonction

Q5. Considérez la déclaration suivante, vue en classe, qui permet de représenter une matrice 2D:

```
// Une matrice 2D
struct Matrix {
  unsigned int r;  // Nombre de lignes
  unsigned int c;  // Nombre de colonnes
  double** values; // Valeurs
};
```

Proposez une implémentation de la fonction

```
bool is_identity(const struct Matrix* m)
```

qui indique si la matrice `m` est une matrice identité. Vous pouvez supposer que `m` est une matrice qui a été construite de façon valide. *Rappel:* une matrice identité est une matrice carrée dont toutes les entrées valent 0, à l'exception des entrées sur la diagonale qui valent 1. *Note:* utilisez le verso au besoin.

Voir solution page suivante

Q1. La séparation modulaire vise à diviser les préoccupations en composantes. La réutilisation d'un module vise à offrir la possibilité de l'utiliser dans différents contextes. L'interchangeabilité modulaire consiste à pouvoir facilement remplacer une composante par une autre dans l'application. C'est donc la spécificité qui correspond à la description ci-haut.

Q2. L'option `-D` de GCC permet de définir un symbole à la compilation. L'option `--symbol` n'existe pas. Les options `-S` et `-d` existent, mais servent à autre chose qui n'a rien à voir avec la définition de symbole.

Q3. Il est important de garder en tête que les macro-fonctions font des substitutions purement textuelles des arguments. En remplaçant successivement les expressions, on obtient ceci:

```
dist(-1, 5 - a)  =  abs(-1 - 5 - a)
                 =  ((-1 - 5 - a) >= 0 ? -1 - 5 - a : -(-1 - 5 - a))
                 =  ((-1 - 5 - 2) >= 0 ? -1 - 5 - 2 : -(-1 - 5 - 2))
                 =  ((-8) >= 0 ? -8 : 8)
                 =  8
```

Q4. Le fichier `module.c` contient l'implémentation du module. En général, la première ligne de ce fichier devrait être `#include "module.h"`. De plus, le fichier `module.c` peut introduire des déclarations de type (en particulier en utilisant `typedef`), des déclarations de constante ou des macros-fonctions sans problème. En revanche, le fichier ne devrait pas contenir de déclaration de fonction publique: celles-ci se trouvent justement dans le fichier d'interface `module.h` (c'est pour cela qu'on utilise le qualificatif *public*). Il est tout de même possible d'inlure des déclarations de fonction, mais elles seront alors privées.

Q5. Tout d'abord, on remarque que si la matrice n'est pas carrée, c'est-à-dire que son nombre de lignes est différent de son nombre de colonnes, alors elle ne peut pas être une matrice identité. Ensuite, on devrait séparer la vérification en deux cas: (1) si on se trouve sur la diagonale, c'est-à-dire que `i == j`, alors on doit forcément avoir `m->values[i][j] == 1` et (2) si on ne se retrouve pas sur la diagonale, c'est-à-dire que `i != j`, alors on doit forcément avoir `m->values[i][j] == 0`. On devrait donc retourner `false` dès qu'une entrée est invalide. Si, à la toute fin, on n'a trouvé aucune entrée invalide, on retourne `true`:

```
bool is_identity(const struct Matrix* m) {
  if (m->r != m->c)
    return false;
  for (int i = 0; i < m->r; ++i)
    for (int j = 0; j < m->c; ++j)
      if ((i == j && m->values[i][j] != 1.0) ||
          (i != j && m->values[i][j] != 0.0))
        return false;
  return true;
}
```

Pour des raisons de lisibilité, on pourrait préférer une version qui teste les deux cas séparément:

```
bool is_identity(const struct Matrix* m) {
  if (m->r != m->c)
    return false;
  for (int i = 0; i < m->r; ++i)
    for (int j = 0; j < m->c; ++j) {
      if (i == j && m->values[i][j] != 1.0)
        return false;
      if (i != j && m->values[i][j] != 0.0)
        return false;
    }
  return true;
}
```