Identification /5

Prénom: .....

Nom: .....

Code permanent: .....

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

Q1. Qu'est-ce qui sera affiché par le code suivant?

```
int arrondir(double x) {
  if (x < 0) return x - 0.5;
  return x + 0.5;
}

int main(void) {
  printf("%d ", arrondir(2.7));
  printf("%d ", arrondir(-2.7));
  printf("%d\n", arrondir(7 / 2));
  return 0;
}
```

- ☐ `3 -3 4`
- ☐ `3 -4 4`
- ■ `3 -3 3`
- ☐ `3 -4 3`

Q2. Lequel des prototypes suivants est valide?

- ☐ `int maximum(int a, b);`
- ☐ `int maximum(int a; int b);`
- ■ `int maximum(int, int);`
- ☐ `maximum(int a, int b);`

Q3. Selon le format exigé dans le cours, que doit contenir la *docstring* de la fonction suivante, en plus de la phrase résumant ce qu'elle fait?

```
void afficher_ligne(char c, int n);
```

- ☐ Un seul `@param` décrivant `c` et `n`
- ■ Un `@param` pour `c` et un pour `n`
- ☐ Un `@param` pour `c`, un pour `n` et un `@return`
- ☐ Un `@return` seulement

Q4. Qu'est-ce qui sera affiché par le code suivant?

```
int soustraire(int n, int k) {
  n = n - k;
  return n;
}

int main(void) {
  int n = 3, k = 4;
  int r = soustraire(k, n);
  printf("%d %d %d\n", n, k, r);
  return 0;
}
```

- ☐ `3 4 -1`
- ☐ `3 1 1`
- ☐ `-1 4 -1`
- ■ `3 4 1`

Q5. Écrivez la définition de la fonction

```
int nombre_diviseurs(int n);
```

précédée de sa *docstring*, dans le format exigé dans le cours. La fonction reçoit un entier `n` supérieur ou égal à 1 et *retourne* le nombre de diviseurs positifs de `n`. Par exemple, `nombre_diviseurs(12)` retourne 6 (les diviseurs sont 1, 2, 3, 4, 6 et 12) et `nombre_diviseurs(7)` retourne 2. La fonction ne doit rien afficher.

Note: utilisez le verso.

Voir solution page suivante

Q1. La fonction `arrondir` retourne un `int`: la valeur de l'expression qui suit `return` est donc convertie en `int`, ce qui *tronque* la partie fractionnaire (arrondi vers zéro). Pour `2.7`, on retourne 3.2, converti en 3; pour `-2.7`, c'est le premier `return` qui est atteint, et il termine immédiatement la fonction: on retourne -3.2, converti en -3, et non en -4. Pour `7 / 2`, l'argument est évalué *avant* l'appel: c'est une division entière, qui vaut 3, converti ensuite en `3.0` pour initialiser le paramètre `x` de type `double`; on retourne 3.5, converti en 3. La sortie est donc `3 -3 3`. Les réponses avec `-4` supposent à tort que la conversion en `int` arrondit vers le bas, et celles qui se terminent par `4` supposent à tort que `7 / 2` vaut 3.5 parce que le paramètre est de type `double`.

Q2. Un prototype contient le type de retour, le nom de la fonction et le type de chaque paramètre, suivis d'un `;`. Les noms des paramètres sont facultatifs: `int maximum(int, int);` est donc valide. Les autres sont invalides: dans `int maximum(int a, b);`, chaque paramètre doit avoir son propre type; les paramètres sont séparés par des virgules, et non des points-virgules; enfin, le type de retour ne peut pas être omis (le compilateur ne suppose plus `int` depuis C99).

Q3. La *docstring* doit décrire *chaque* paramètre avec son propre `@param`: il en faut donc un pour `c` et un pour `n`. Comme la fonction est `void`, elle ne retourne aucune valeur: il ne faut pas de `@return`.

Q4. Les arguments sont associés aux paramètres *dans l'ordre*, et non selon leur nom: le paramètre `n` reçoit la valeur de `k`, soit 4, et le paramètre `k` reçoit celle de `n`, soit 3. La fonction calcule donc $4 - 3 = 1$, et `r` vaut 1. De plus, le passage se fait *par valeur*: les paramètres sont des copies, distinctes des variables de `main` malgré leurs noms identiques, et modifier le paramètre `n` ne modifie pas les variables de `main`, qui valent toujours 3 et 4, d'où la sortie `3 4 1`. Les réponses `3 4 -1` et `-1 4 -1` supposent à tort que les arguments sont associés selon leur nom, et `-1 4 -1` et `3 1 1` supposent à tort que la modification du paramètre est répercutée sur la variable de `main` correspondante.

Q5. La *docstring* commence par une phrase résumant ce que fait la fonction, suivie d'un paragraphe précisant la valeur admise pour `n`, puis d'un `@param` et d'un `@return`, puisque la fonction retourne une valeur. Pour compter les diviseurs, on parcourt tous les entiers `d` de 1 à `n` avec une boucle `for`, et on incrémente un compteur lorsque `d` divise `n`, c'est-à-dire lorsque `n % d == 0`. Le résultat est *retourné* avec `return` et non affiché avec `printf`: c'est à l'appelant de décider quoi en faire (l'afficher, le comparer, etc.).

```
/**
 * Calcule le nombre de diviseurs positifs d'un entier.
 *
 * L'entier n doit etre superieur ou egal a 1.
 *
 * @param n  L'entier dont on compte les diviseurs
 * @return   Le nombre d'entiers d entre 1 et n tels que d divise n
 */
int nombre_diviseurs(int n) {
  int nb = 0;
  for (int d = 1; d <= n; d++)
    if (n % d == 0)
      nb++;
  return nb;
}
```