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?

```
void a(void) {
  printf("A");
}

void b(void) {
  printf("B");
  a();
  printf("b");
}

int main(void) {
  printf("M");
  b();
  a();
  printf("\n");
  return 0;
}
```

- ☐ `MBbAA`
- ■ `MBAbA`
- ☐ `MBAA`
- ☐ `ABAbMBAbA`

Q2. Quelle variable a une *portée de bloc*, mais une *durée de vie statique*?

- ☐ Une variable locale déclarée sans `static`
- ☐ Une variable globale
- ☐ Une zone de mémoire réservée avec `malloc`
- ■ Une variable locale déclarée `static`

Q3. Qu'est-ce qui sera affiché par le code suivant?

```
int total = 0;
for (int i = 1; i <= 3; i++) {
  int s = 10;
  s += i;
  total += s;
}
printf("%d\n", total);
```

- ☐ `13`
- ☐ `33`
- ■ `36`
- ☐ `40`

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

```
int n = 1;

void doubler(void) {
  n = 2 * n;
}

void afficher(void) {
  printf("%d ", n);
}

int main(void) {
  int n = 5;
  doubler();
  {
    int n = 7;
    doubler();
    afficher();
    printf("%d ", n);
  }
  printf("%d\n", n);
  return 0;
}
```

- ■ `4 7 5`
- ☐ `7 7 5`
- ☐ `4 7 7`
- ☐ `14 14 10`

Q5. Complétez la fonction suivante:

```
void afficher_cadre(void) {
  const int largeur = 5;
  const int hauteur = 3;
  // A completer
}
```

Elle doit afficher un cadre de `hauteur` lignes et de `largeur` colonnes: les caractères du bord sont des `#`, ceux de l'intérieur sont des `-`, et chaque ligne se termine par un saut de ligne. Avec les valeurs ci-dessus, la fonction affiche

```
#####
#---#
#####
```

Votre code doit utiliser les constantes `largeur` et `hauteur`, sans les modifier, et fonctionner pour toutes valeurs supérieures ou égales à 2.

Note: utilisez le verso.

Voir solution page suivante

Q1. Le programme commence toujours par `main`, peu importe l'ordre des fonctions dans le fichier: on affiche d'abord `M`. L'appel `b()` affiche `B`, puis appelle `a`, qui affiche `A`; à la fin de `a`, l'exécution revient *juste après l'appel*, dans `b`, qui affiche `b` avant de revenir dans `main`. Enfin, `a()` affiche un dernier `A`, d'où la sortie `MBAbA`. Les réponses `MBbAA`, `MBAA` et `ABAbMBAbA` supposent respectivement à tort qu'un appel est reporté à la fin de la fonction appelante, que la fin de `a` ramène directement dans `main` et que les fonctions s'exécutent avant `main`, dans l'ordre où elles sont écrites.

Q2. Une variable locale déclarée `static` n'est visible que dans son bloc (portée de bloc), mais elle existe pendant toute l'exécution du programme (durée de vie statique): les deux notions sont indépendantes. Une variable locale ordinaire a une durée de vie automatique, une variable globale a une portée de fichier, et une zone réservée avec `malloc` a une durée de vie dynamique et n'a pas de nom, donc pas de portée.

Q3. La variable `s` est déclarée dans le corps de la boucle: sa durée de vie est *automatique*. Elle est donc créée à chaque entrée dans le bloc, c'est-à-dire à chaque itération, et réinitialisée à 10: la valeur de l'itération précédente n'est pas conservée. On ajoute donc 11, 12 et 13 à `total`, déclarée hors de la boucle, soit 36. Les réponses `13`, `33` et `40` correspondent respectivement à une confusion entre `total` et `s`, à une erreur sur les bornes de la boucle ($10+11+12$) et à l'idée erronée que `s` conserve sa valeur ($11+13+16$).

Q4. Les fonctions `doubler` et `afficher` sont définies hors de `main`: elles ne voient pas les variables locales de `main` et utilisent toujours la variable *globale* `n`. Les deux appels à `doubler` la font passer de 1 à 2, puis à 4, et `afficher` affiche donc `4`. Dans `main`, la variable de bloc (valant 7) masque la locale (valant 5), qui masque elle-même la globale: le premier `printf` affiche `7`. À la sortie du bloc, la locale redevient visible: seulement masquée, et non modifiée, elle vaut toujours 5, d'où la sortie `4 7 5`. La réponse `4 7 7` suppose à tort que la variable masquée est remplacée par celle du bloc; `7 7 5` et `14 14 10` supposent à tort qu'une fonction voit les variables de la fonction qui l'appelle (pour `afficher` seulement, ou pour les deux).

Q5. Il faut parcourir le cadre ligne par ligne, puis, dans chaque ligne, colonne par colonne: on utilise donc deux boucles `for` imbriquées, la boucle externe sur les lignes `i` et la boucle interne sur les colonnes `j`. Un caractère est sur le bord s'il est dans la première ou la dernière ligne (`i == 0` ou `i == hauteur - 1`), ou dans la première ou la dernière colonne (`j == 0` ou `j == largeur - 1`): un `if`/`else` choisit alors entre `#` et `-`. Le saut de ligne est affiché *après* la boucle interne, une seule fois par ligne, et non à chaque caractère. Les bornes des boucles et les conditions doivent utiliser `largeur` et `hauteur` plutôt que les valeurs 5 et 3, et les compteurs `i` et `j` sont déclarés dans le plus petit bloc où ils sont utilisés. Comme le `if`/`else` ne fait que choisir le caractère à afficher, on peut aussi le remplacer par l'opérateur ternaire, en conservant le résultat de la condition dans une variable `bord` pour la lisibilité (à droite):

```
void afficher_cadre(void) {
  const int largeur = 5;
  const int hauteur = 3;
  for (int i = 0; i < hauteur; i++) {
    for (int j = 0; j < largeur; j++) {
      if (i == 0 || i == hauteur - 1 ||
          j == 0 || j == largeur - 1) {
        printf("#");
      } else {
        printf("-");
      }
    }
    printf("\n");
  }
}
```

```
void afficher_cadre(void) {
  const int largeur = 5;
  const int hauteur = 3;
  for (int i = 0; i < hauteur; i++) {
    for (int j = 0; j < largeur; j++) {
      int bord = i == 0 || j == 0 ||
                 i == hauteur - 1 ||
                 j == largeur - 1;
      printf("%c", bord ? '#' : '-');
    }
    printf("\n");
  }
}
```