

Exercice 1

1. Entourez les déclarations correctes:
 - a. `int x[4] = {8, 7, 6, 4, 3};`
 - b. `int y[] = {8, 7, 6, 4, 3};`
 - c. `const int TAILLE = 4;`
 - d. `int z[TAILLE];`
 - e. `float 2x = 3.5;`
 - f. `bool ok = true;`
 2. Pour chaque ligne, indiquez: la valeur après exécution et si une erreur peut survenir.
 - `int a = 5;`
 - `double b = 5 / 2;`
 - `char c = 'A' + 3;`
 - `bool d = (7 > 3) && (2 < 1)`
 3. Que renvoie les expressions suivantes :
 - `(5 > 3) || (2 > 7)?`
 - `!(4 <= 2) && (3 + 2 > 4)`
 4. Si on a les déclarations suivantes :


```
void triple(int& entier) {entier = 3 * entier; }
int a[3] = (4, 5, 6), nombre = 2;
```

 Quels sont les appels de fonctions acceptables ?
 - a. `triple (a[2]);`
 - b. `triple (a[3]);`
 - c. `triple (a[nombre]);`
 - d. `triple (a);`
 - e. `triple (nombre);`
 5. Donnez seulement le prototype de la fonction trouver qui ne retourne rien et qui reçoit comme paramètres un tableau d'entiers, le nombre d'éléments dans le tableau et la valeur maximale du tableau qui est modifié dans la fonction.
Réponse:
 6. Donnez le prototype et le corps d'une fonction echanger qui reçoit deux entiers par référence et échange leurs valeurs.
Réponse:
- Donnez ensuite un exemple d'appel de cette fonction.
Réponse:

Corrigé :

1. Déclarations correctes:
 - a. `int x[4] = {8, 7, 6, 4, 3};` ✗ — 5 valeurs pour un tableau de taille 4
 - b. `int y[] = {8, 7, 6, 4, 3};` ✓ — taille déduite automatiquement
 - c. `const int TAILLE = 4;` ✓
 - d. `int z[TAILLE];` ✓ — valide si TAILLE est une constante de compilation
 - e. `float 2x = 3.5;` ✗ — identifiant invalide (commence par un chiffre)
 - f. `bool ok = true;` ✓

2. La valeur après exécution et si une erreur peut survenir

Instruction	Valeur	Erreur possible
<code>int a = 5;</code>	<code>a = 5</code>	Non
<code>double b = 5 / 2;</code>	<code>b = 2.0</code>	Non
<code>char c = 'A' + 3;</code>	<code>c = 'D' (ASCII 68)</code>	Non
<code>bool d = (7 > 3) && (2 < 1)</code>	<code>d = false</code>	Non

3. Expressions booléennes :

`(5 > 3) || (2 > 7) → true || false → 1 || false → true`

`!(4 <= 2) && (3 + 2 > 4) → !false && true → true && true → true`

4. Appels acceptables de `triple(int& entier)`. Avec : `int a[3] = {4, 5, 6}`, `nombre = 2` ;

- a. `triple(a[2]);` ✓ — `a[2]` est un `int` modifiable (*lvalue*)
- b. `triple(a[3]);` ✗ — hors bornes (accès mémoire indéfini)
- c. `triple(a[nombre]);` ✓ — `a[2]`, *lvalue* valide
- d. `triple(a);` ✗ — `a` est un tableau, pas un `int`
- e. `triple(nombre);` ✓ — `nombre` est un `int` modifiable

5. Prototype de la fonction `trouver` :

`void trouver(int tableau[], int taille, int& max);`

6. Fonction `echanger` :

```
// Prototype
void echanger(int& a, int& b);

// Corps
void echanger(int& a, int& b) {
    int temp = a;
    a = b;
    b = temp;
}

// Exemple d'appel
int x = 3, y = 7;
echanger(x, y); // x vaut 7, y vaut 3
```

Exercice 2

Soient les déclarations suivantes :

```
int tableau[10] = {1, 3, -2, 10, 4, 9, -5, 15, 6, 5};
int i, somme = 0;
```

Pour les boucles de répétition suivantes, trouvez l'initialisation, l'expression booléenne et l'actualisation.

```
for (initialisation, expression booléenne, actualisation)
    cout << tableau[i] << endl; }
```

a) Affichez tous les éléments du tableau.

Réponse :

b) Affichez le contenu des éléments du tableau aux indices pairs du tableau.

Réponse:

c) Affichez le contenu des éléments du tableau tant que leur somme ne dépasse par la valeur 15 et le nombre d'éléments du tableau.

Réponse:

Corrigé :

a)

```
for (i = 0; i < 10; i++)
```

→ *init: i = 0 | booléenne: i < 10 | actualisation: i ++*

b)

```
for (i = 0; i < 10; i += 2)
```

→ *init: i = 0 | booléenne: i < 10 | actualisation: i += 2 Affiche : 1, - 2, 4, - 5, 6*

c)

```
for (i = 0; i < 10 && somme <= 15; i++)
```

→ *init: i = 0, somme = 0 | booléenne: i < 10 && somme <= 15 | actualisation: i ++ (avec somme += tableau[i] dans le corps)*

Simulation : somme: 0→1→4→2→12→16 (stop à i = 4 avant affichage de 4) Affiche : 1, 3, - 2, 10

Exercice 3

1. Quel sera l'affichage obtenu après l'exécution des instructions suivantes ?

<pre>for (int i = 0; i < 10; i++) { int j; for(j = 0; j < (1&2); j++) { i++; } cout << i*j <<< " "; }</pre>	Réponse
--	---------

2. Quel sera l'affichage obtenu après l'exécution des instructions suivantes?

<pre>bool c = true; int a = 7; while(a < 10 && c) { switch(a){ case 0; case 1: a += 2; break; case 2: c = false; case 5: a = a + 3; break; case 7: a += (a / 2); break; case 9: a = 0; break; default: a++; } cout << a << " "; }</pre>	Réponse
--	---------

3. Donnez l'affichage du programme suivant at l'utilisateur entre le mot << genie >>

<pre>#include <iostream> #include <string> using namespace std; int main(){ string mot; int i, j; cout<<"Entrez un mot: "; cin>> mot; for (i = 0; i < mot.size(); i++) { for (j = 0; j <= 1; j++) cout << mot[j]; cout << endl; } return 0; }</pre>	Réponse
---	---------

4. Corriger les erreurs dans le programme suivant

<pre>#include <iostream> using namespace std; int main() { for (i = 1; i <= 5; i++) if (i % 2 == 0) cout << i << endl; }</pre>	Programme corrigé
---	-------------------

Corrigé :

1.

<pre>for (int i = 0; i < 10; i++) { int j; for(j = 0; j < (1&2); j++) { i++; } cout << i*j <<< " "; }</pre>	Réponse : $1 \& 2$ (AND bit à bit) = $01 \& 10 = 0$ La boucle interne ne s'exécute jamais ($j < 0$ est faux dès le départ), donc $j = 0$ à chaque fois. $i * j = i * 0 = 0$ à chaque itération. Affichage : 0 0 0 0 0 0 0 0 0 0
--	--

2.

<pre>bool c = true; int a = 7; while(a < 10 && c) { switch(a){ case 0; case 1: a += 2; break; case 2: c = false; case 5: a = a + 3; break;</pre>	Réponse : On a : $c = \text{true}$, $a = 7$ Action (case 7) : Division entière de $7/2 = 3$. On l'ajoute à a ($7 + 3$), donc a devient 10. Affichage : Le programme écrit 10 Sortie : a vaut 10, la condition $a < 10$ devient fausse,
---	---

```

        case 7: a += (a / 2);
                break;
        case 9: a = 0;
                break;
        default: a++;
    }
    cout << a <<< " ";
}

```

la boucle s'arrête.
Affichage final : 10

3.

```

#include <iostream>
#include <string>
using namespace std;
int main(){
    string mot;
    int i, j;
    cout<<"Entrez un mot: ";
    cin>> mot;
    for (i = 0; i < mot.size(); i++)
    {
        for (j = 0; j <= 1; J++)
            cout << mot[j]; cout << endl;
    }
    return 0;
}

```

Réponse :
Pour "genie" (taille 5) :

g
ge
gen
geni
genie

4.

```

#include <iostream>
using namespace std;
int main() {
    for (i = 1; i <= 5; i++)
        if (i % 2 == 0)
            cout << i << endl
}

```

Programme corrigé :

```

int main() {
    for (int i = 1; i <= 5; i++)
        if (i % 2 == 0)
            cout << i << endl;
}

```

Exercice 4

Écrire un programme qui :

1. Demande à l'utilisateur un entier positif n.
2. Calcule la somme de tous les chiffres de n.
3. Affiche le résultat.

Exemple :

Entrer un entier positif : 12345

La somme des chiffres est: 15

Note: Il est inutile d'écrire des commentaires et de mettre les include au début du programme.

Corrigé :

```
#include <iostream>
using namespace std;
int main() {
    int n, somme = 0;
    cout << "Entrer un entier positif : ";
    cin >> n;
    while (n > 0) {
        somme += n % 10;
        n /= 10;
    }
    cout << "La somme des chiffres est : " << somme << endl;
    return 0;
}
```

