

Identification

/5

Prénom:

Nom:

Code permanent:

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

Q1. Quelle norme du langage C a été publiée en 1989?

- | | |
|--|----------------------------------|
| ☒ ANSI-C | ☐ ISO C11 |
| ☐ ISO C99 | ☐ ISO C23 |

Q2. Qu'est-ce qui sera affiché sur la sortie standard par le programme suivant?

```
#include <stdio.h>
#include <stdbool.h>

enum Etat { OK, ERREUR };

int main(void) {
    char c = '0';
    int i = 010;
    bool b = OK;
    enum Etat e = ERREUR;
    if (c) printf("c ");
    if (i) printf("i ");
    if (b) printf("b ");
    if (e) printf("e ");
    return 0;
}
```

- | | |
|---|------------------------------|
| ☒ c i e | ☐ c e |
| ☐ c i b e | ☐ i e |

Q3. Considérez un répertoire courant contenant uniquement le fichier `prog.c`. Quel fichier sera produit par la commande suivante?

```
$ gcc -c prog.c
```

- | | |
|--------------------------------|--|
| ☐ a.out | ☒ prog.o |
| ☐ prog | ☐ prog.exe |

Q4. Parmi les affirmations suivantes à propos de Git, laquelle est vraie?

- ☐ La commande `git init` permet de créer une copie locale d'un dépôt distant existant
- ☐ La commande `git commit` permet de téléverser l'état courant du projet vers un dépôt distant
- ☒ La commande `git add` permet d'ajouter des changements à l'aire de *staging* en vue du prochain *commit*
- ☐ La commande `git log` permet de récupérer les changements effectués par d'autres personnes sur un dépôt distant

Q5. Écrivez un programme complet en C qui lit des caractères sur l'entrée standard, un à un, jusqu'à la fin du fichier (EOF), et qui affiche, une fois la lecture terminée, le nombre total de voyelles minuscules (a, e, i, o, u) qui ont été lues. *Rappel:* la fonction `int getchar(void)` lit un caractère sur l'entrée standard et retourne sa valeur, convertie en `int`, ou la constante `EOF` lorsque la fin du fichier est atteinte. *Note:* utilisez le verso au besoin.

[Voir explications page suivante](#)

Q1. Le comité de normalisation ANSI a publié la norme ANSI-C en 1989, ce qui en fait la toute première norme du langage C. Les normes ISO C99, ISO C11 et ISO C23 sont des révisions plus récentes du standard, publiées respectivement en 1999, 2011 et 2024.

Q2. En C, la valeur 0 est considérée comme fausse, alors que toute autre valeur est considérée comme vraie. La variable `c` contient le caractère `'o'`, dont le code ASCII (48) est non nul: il ne faut pas confondre le caractère `'o'` avec la valeur entière 0. La condition est donc vraie et `c` est affiché. La variable `i` est initialisée avec le littéral octal `010`, qui vaut 8 en décimal (et non 10), donc non nulle: `i` est affiché. La variable `b`, de type `bool`, est initialisée avec `OK`, la première constante de l'énumération, qui vaut 0: la condition est donc fausse et rien n'est affiché pour elle. La variable `e` vaut `ERREUR`, la seconde constante de l'énumération, qui vaut 1, donc non nulle: `e` est affiché. On obtient donc, dans l'ordre, `c i e`.

Q3. L'option `-c` de `gcc` demande d'effectuer uniquement la compilation, sans l'édition des liens. Le fichier source `prog.c` est donc traduit en un fichier objet, dont le nom est obtenu en remplaçant l'extension `.c` par `.o`, soit `prog.o`. Le fichier `a.out` aurait plutôt été produit si on avait combiné compilation et édition des liens (par exemple avec `gcc prog.c`), et `prog` n'aurait été obtenu qu'en spécifiant explicitement le nom de l'exécutable, par exemple avec `gcc -o prog prog.c`. L'extension `.exe` n'est quant à elle utilisée que sous Windows.

Q4. La commande `git init` sert à créer un *nouveau* dépôt vide, et non à cloner un dépôt distant existant; c'est plutôt le rôle de `git clone`. La commande `git commit` sauvegarde l'état courant de l'aire de *staging* dans l'historique *local* du dépôt; elle ne communique avec aucun dépôt distant, contrairement à `git push`, qui sert à y téléverser les changements. La commande `git add` permet effectivement d'ajouter des changements à l'aire de *staging*, en vue du prochain *commit*: c'est donc l'énoncé vrai. Finalement, la commande `git log` permet de consulter l'historique des *commits* déjà enregistrés localement; pour récupérer les changements effectués par d'autres personnes sur un dépôt distant, il faut plutôt utiliser `git pull`.

Q5. La variable qui reçoit la valeur retournée par `getchar` doit être de type `int` (et non `char`), car `EOF` est une valeur entière négative qui ne peut pas nécessairement être représentée dans un `char`. On relit un caractère à la fois à l'aide de `getchar` jusqu'à ce que la valeur `EOF` soit obtenue, tout comme dans l'exemple `maj.c` vu en classe. Pour chaque caractère lu, on vérifie s'il s'agit d'une voyelle minuscule et, le cas échéant, on incrémente un compteur. Une fois la boucle terminée, on affiche la valeur finale du compteur. Deux solutions équivalentes sont présentées ci-dessous: la première utilise une instruction `if`, alors que la seconde utilise une instruction `switch`. Une troisième solution, moins bonne, utilisant une chaîne de `if/else if`, est présentée ensuite.

Version avec if

```
#include <stdio.h>

int main(void) {
    int c;
    int nb_voyelles = 0;
    while ((c = getchar()) != EOF) {
        if (c == 'a' || c == 'e' || c == 'i'
            || c == 'o' || c == 'u') {
            nb_voyelles++;
        }
    }
    printf("%d\n", nb_voyelles);
    return 0;
}
```

Version avec switch

```
#include <stdio.h>

int main(void) {
    int c;
    int nb_voyelles = 0;
    while ((c = getchar()) != EOF) {
        switch (c) {
            case 'a': case 'e': case 'i':
            case 'o': case 'u':
                nb_voyelles++;
        }
    }
    printf("%d\n", nb_voyelles);
    return 0;
}
```

Version avec if. La condition combine cinq comparaisons à l'aide de l'opérateur `||`: elle est vraie dès que l'une d'entre elles est vraie. Il faut répéter `c ==` dans chacune des comparaisons: une erreur fréquente consiste à écrire `c == 'a' || 'e' || ...`, une condition qui est *toujours* vraie, puisque `'e'` est une valeur non nulle (voir la question 2). Il faut aussi prendre garde à ne pas confondre l'opérateur de comparaison `==` avec l'opérateur d'affectation `=`.

Version avec switch. L'expression `c` n'est écrite qu'une seule fois, puis comparée à chacune des constantes indiquées par les `case`. Les cinq `case` correspondant aux voyelles minuscules sont regroupés, sans `break` entre eux, de sorte qu'ils partagent tous la même instruction, qui incrémente le compteur. Aucun `break` n'est nécessaire après cette instruction, puisqu'il s'agit du dernier (et seul) groupe de `case` du `switch`: une fois le compteur incrémenté, l'exécution atteint de toute façon la fin du `switch`. Si le caractère lu ne correspond à aucune des cinq voyelles, aucun `case` ne s'applique et rien ne se produit.

Version avec if/else if. On pourrait aussi utiliser une chaîne de `if/else if`, avec une branche par voyelle, comme ci-dessous. Cette solution est correcte, mais elle est moins bonne que les deux précédentes, car elle est beaucoup moins concise: l'instruction `nb_voyelles++` doit être répétée dans chacune des cinq branches, ce qui allonge le code et augmente les risques d'erreur lorsqu'on le modifie.

```
#include <stdio.h>

int main(void) {
    int c;
    int nb_voyelles = 0;
    while ((c = getchar()) != EOF) {
        if (c == 'a') {
            nb_voyelles++;
        } else if (c == 'e') {
            nb_voyelles++;
        } else if (c == 'i') {
            nb_voyelles++;
        } else if (c == 'o') {
            nb_voyelles++;
        } else if (c == 'u') {
            nb_voyelles++;
        }
    }
    printf("%d\n", nb_voyelles);
    return 0;
}
```

Comparaison. Les trois versions sont correctes et produisent exactement le même résultat, mais la chaîne de `if/else if` est à éviter, pour les raisons mentionnées plus haut. Le `switch` est bien adapté ici, puisqu'on compare une même expression entière à plusieurs constantes: le code est plus concis, et il est facile d'ajouter ou de retirer une voyelle. En revanche, le `switch` est moins général que le `if`: chaque `case` doit être une expression *constante* entière, de sorte qu'il ne permet pas, par exemple, de tester un intervalle (comme `c >= 'a' && c <= 'z'`) ou de comparer à la valeur d'une autre variable. Il comporte aussi un piège: si on ajoutait d'autres groupes de `case` (par exemple pour compter aussi les consonnes), il faudrait penser à terminer chaque groupe par un `break`, sans quoi l'exécution se poursuivrait dans le groupe suivant.