ALGORITHMIQUE · CHAPITRE 15 · CONDITIONS ET BOUCLES

Répéter TANT QUE

Suivez une boucle while pas à pas, y compris si elle ne s'exécute jamais. Comprenez l'initialisation, la progression et le risque de boucle infinie.

Par Ali ZAINOUL · Cours gratuit

Ce que vous allez apprendre

- Identifier initialisation, test, corps et mise à jour.
- Expliquer pourquoi le corps peut ne jamais s'exécuter.
- Repérer une progression et comprendre une sentinelle.

Prérequis. Comprendre un tour de boucle et les conditions booléennes.

Quand faut-il arrêter le compte à rebours ?

Le nombre `remaining` représente des unités encore à traiter. Il commence à 3. Tant qu'il est strictement positif, on l'affiche puis on le diminue de 1. Nous travaillons avec un entier non négatif : cette convention est importante pour justifier l'arrêt.

Les trois moments d'une boucle while

`WHILE` signifie "tant que". Cette boucle commence par évaluer une condition ; si elle est vraie, elle exécute le corps puis revient au test. Si elle est fausse, elle continue après la boucle. Notre pseudo-code écrit `WHILE ... DO` et `END WHILE`. Python et MATLAB utilisent le mot-clé `while`.

L'**initialisation** fixe `remaining` à 3 avant le premier test. La **condition de poursuite** est `remaining > 0`. La **mise à jour** soustrait 1 après l'affichage. Une **boucle infinie** est une boucle dont le déroulement ne rejoint jamais la sortie ; oublier la diminution peut en créer une pour une

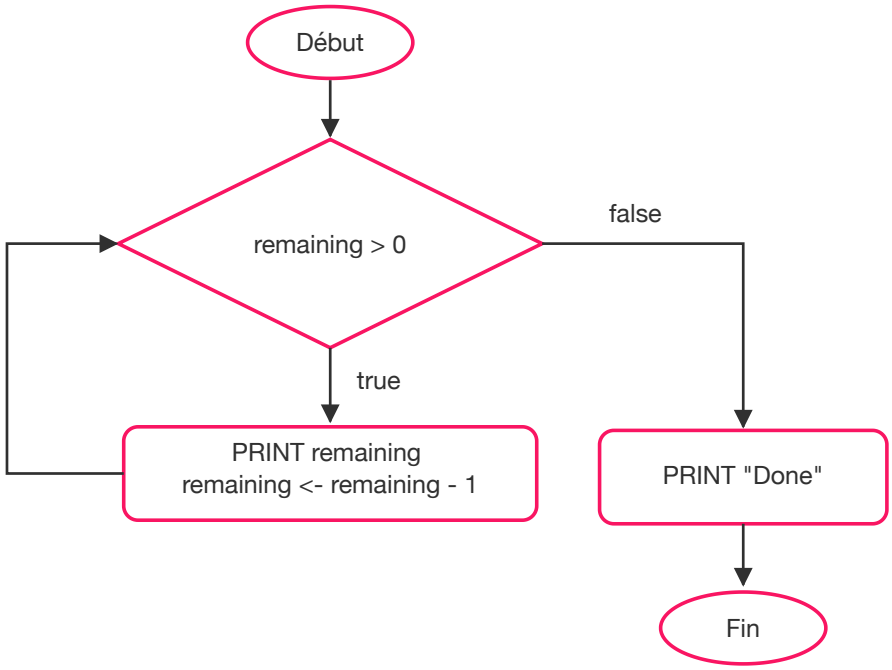
valeur de départ positive.

Pour une entrée de 3, les tests observent successivement 3, 2, 1 puis 0. Les trois premiers sont vrais ; le dernier est faux et n'exécute pas le corps. Il y a donc trois tours mais quatre évaluations de la condition. Pour une entrée de 0, le premier test est déjà faux : aucun affichage de décompte n'a lieu.

Notre entrée autorisée est un entier non négatif. Une soustraction de 1 conserve des entiers et rejoint zéro après un nombre fini de tours. Cette explication dépend du domaine choisi et de la mise à jour. Elle ne prouve rien sur une autre boucle qui attendrait un événement extérieur.

Le test précède chaque tour

Après le corps, on revient au test avec la nouvelle valeur de remaining.



▼ **Lire les parcours sans animation**

remaining = 3

1. Début du parcours avec remaining = 3. Les données sont déjà disponibles.

2. Test 1 avec remaining = 3 : remaining > 0 donne true (vrai).
3. Action choisie : PRINT remaining; remaining <- remaining - 1.
4. Test 2 avec remaining = 2 : remaining > 0 donne true (vrai).
5. Action choisie : PRINT remaining; remaining <- remaining - 1.
6. Test 3 avec remaining = 1 : remaining > 0 donne true (vrai).
7. Action choisie : PRINT remaining; remaining <- remaining - 1.
8. Test 4 avec remaining = 0 : remaining > 0 donne false (faux).
9. Action choisie : PRINT "Done".
10. Fin du parcours. Résultat observable : 3, 2, 1, Done.

remaining = 0

1. Début du parcours avec remaining = 0. Les données sont déjà disponibles.
2. Test 1 avec remaining = 0 : remaining > 0 donne false (faux).
3. Action choisie : PRINT "Done".
4. Fin du parcours. Résultat observable : Done.

TANT QUE ne signifie pas "faire au moins une fois". Le test est effectué avant le premier tour, puis de nouveau après chaque exécution du corps. Une entrée initiale égale à zéro saute donc directement la répétition.

Faire évoluer ce que le test observe

Pseudo-code

```
// example.pseudo
// countdown
remaining <- 3
WHILE remaining > 0 DO
    PRINT remaining
    remaining <- remaining - 1
END WHILE
PRINT "Done"
```

Python

```python
# countdown.py
remaining = 3
while remaining > 0:
    print(remaining)
    remaining = remaining - 1
print("Done")
```

MATLAB

```
% countdown.m
remaining = 3;
while remaining > 0
    disp(remaining);
    remaining = remaining - 1;
end
disp('Done');
```

La mise à jour change la variable utilisée par le test. Supprimer la diminution, ou modifier une autre variable à sa place, rendrait ici la condition toujours vraie. Une boucle ne s'arrête pas simplement parce que son corps a déjà été exécuté plusieurs fois.

Lire le test à chaque retour

Trace du compte à rebours

remaining avant le test	Test > 0	Action
3	vrai	Afficher 3, puis passer à 2
2	vrai	Afficher 2, puis passer à 1
1	vrai	Afficher 1, puis passer à 0
0	faux	Ne plus entrer dans le corps

Il y a trois tours mais quatre évaluations du test. La dernière évaluation sert à constater qu'on ne doit plus recommencer. Sur ces entrées, remaining est entier, diminue de 1 à chaque tour et ne peut pas rester strictement positif indéfiniment.

Quand l’arrêt dépend d’une valeur spéciale

Une **sentinelle** est une valeur qui signale la fin, par exemple la commande "stop". Dans ce cas, le corps doit obtenir une nouvelle commande avant de revenir au test. La sentinelle elle-même ne doit pas être traitée comme une commande ordinaire. Si une source externe ne fournit jamais "stop", la terminaison n'est pas garantie par cette règle seule.

Détecter une boucle incorrecte

▼ Pourquoi remaining ← remaining + 1 empêcherait-il l'arrêt à partir de 3 ?

La valeur resterait strictement positive et s'éloignerait de zéro. Le corps continuerait donc à être exécuté. Il manque une progression vers une situation où le test devient faux.

▼ Combien de tours avec remaining = 1 ? Et avec remaining = 0 ?

Un tour dans le premier cas : 1 est affiché, puis remaining devient 0. Zéro tour dans le second cas : le premier test est déjà faux. Le message placé après la boucle apparaît dans les deux cas.

Justifier l'arrêt sur les entrées autorisées

Une explication complète nomme la variable qui progresse, son évolution et la condition qui met fin à la répétition. La phrase "cette boucle termine" doit toujours être reliée aux entrées et aux mises à jour effectivement prévues.

À retenir - fiche mémo

Avant

Initialisez les valeurs nécessaires au premier test.

Pendant

WHILE teste avant chaque tour ; le corps met à jour l'état.

Arrêt

Avec remaining entier non négatif et une diminution de 1, le décompte atteint zéro.

Pseudo-code

```
// example.pseudo
// countdown
WHILE remaining > 0 DO
    PRINT remaining
    remaining <- remaining - 1
END WHILE
```

Point de vigilance. Trois tours de ce décompte nécessitent quatre tests, dont le dernier est faux.

Consulter la cheatsheet : Conditions et boucles · **Télécharger la fiche du thème en PDF**

Version web : https://www.clearcode.fr/courses/programming/algorithmic/control-flow/loops/while