ALGORITHMIQUE · CHAPITRE 24 · FONCTIONS

Suivre les appels et les retours

Tracez sum_to(3) avec une animation et des affichages Call/Return. Distinguez les appels en attente, leurs variables locales et les résultats renvoyés.

Par Ali ZAINOUL · Cours gratuit

Ce que vous allez apprendre

- Séparer la descente des appels et la remontée des résultats.
- Identifier l'état local de chaque appel.
- Lire une trace produite par une fonction instrumentée.

Prérequis. Connaître le cas de base et la progression de sum_to.

Où le calcul 3 + … est-il conservé ?

Lorsque `sum_to(3)` appelle la fonction pour 2, le calcul initial n'est pas terminé. L'exécution conserve les informations nécessaires pour reprendre ensuite l'addition avec 3. Il faut donc distinguer l'appel en cours des appels qui attendent son résultat.

Un état local par appel

Les appels pour 3, 2, 1 et 0 possèdent chacun leur valeur locale de n. Le nom est le même, mais il ne désigne pas une seule variable que tous les appels remplaceraient à tour de rôle. À son retour, l'appel pour 2 utilise encore son propre n = 2 pour effectuer son addition.

Prévoir la prochaine étape

PRÉVOIR LE PROCHAIN APPEL OU RETOUR

Étape courante : `sum_to(3)`

Calcul en attente : `3 + sum_to(2)`

3 est différent de 0. Cet appel attend le résultat pour 2.

▼ **Lire tous les appels et les retours sans animation**

1. `sum_to(3)` : 3 est différent de 0. Cet appel attend le résultat pour 2. `3 + sum_to(2)`.
2. `sum_to(2)` : L'appel pour 2 attend celui pour 1. Le premier appel reste en attente. `3 + (2 + sum_to(1))`.
3. `sum_to(1)` : L'appel pour 1 attend celui pour 0. `3 + (2 + (1 + sum_to(0)))`.
4. `sum_to(0)` : Le cas de base est atteint : cet appel renvoie 0 sans nouvel appel récursif. `3 + (2 + (1 + 0))`.
5. `Return from sum_to(1)` : L'appel pour 1 calcule 1 + 0 et renvoie 1. `3 + (2 + 1)`.
6. `Return from sum_to(2)` : L'appel pour 2 reçoit 1, calcule 2 + 1 et renvoie 3. `3 + 3`.
7. `Return from sum_to(3)` : L'appel initial reçoit 3, calcule 3 + 3 et renvoie 6. Aucun appel ne reste en attente. `6`.

Cette visualisation montre sept moments du calcul pour n = 3. Le moment du cas de base rassemble l'arrivée dans l'appel pour 0 et sa réponse immédiate. L'exécution comporte quatre appels et quatre retours ; certains moments réunissent donc plusieurs événements.

Comprendre ce qui reste en attente

Une **pile d'appels** représente les appels commencés qui n'ont pas encore terminé. Dans notre lecture simplifiée, le dernier appel commencé est le premier à pouvoir rendre son résultat. Chaque appel conserve son propre paramètre n et l'endroit où le calcul doit reprendre. Cette pile explique l'ordre des retours ; elle ne signifie pas que les additions sont toutes effectuées au départ.

Une fonction **instrumentée** contient des observations ajoutées pour comprendre son exécution. Nos messages anglais `"Call"` et `"Return"` signalent respectivement l'entrée dans un appel et son retour. Ils ne changent pas la somme calculée. Ils ajoutent en revanche des affichages, donc un travail qui n'existerait pas dans la version silencieuse.

Dans `3 + sum_to(2)`, l'addition de 3 attend le résultat du second terme. Lorsque `sum_to(2)` renvoie 3, l'appel initial peut calculer 6. Les valeurs n égales à 3 et à 2 ne sont pas deux écritures simultanées de la même variable globale : elles appartiennent à deux appels distincts.

Dans l'animation, le cas de base et son retour immédiat sont réunis dans une étape. Les autres étapes distinguent la descente et la remontée. Utilisez les boutons pour prévoir la prochaine ligne, puis expliquez quelles additions restent en attente. Lire seulement la dernière ligne masque précisément le mécanisme à comprendre.

Rendre les appels observables

Pseudo-code

```
// example.pseudo
// sum_to_traced
FUNCTION sum_to_traced(n)
    PRINT "Call", n
    IF n == 0 THEN
        result <- 0
    ELSE
        result <- n + sum_to_traced(n - 1)
    END IF
    PRINT "Return", n, result
    RETURN result
END FUNCTION
```

Python

```python
# sum_to_traced.py
def sum_to_traced(n):
    print("Call", n)
    if n == 0:
        result = 0
    else:
        result = n + sum_to_traced(n - 1)
    print("Return", n, result)
    return result

result = sum_to_traced(3)
print(result)  # 6
```

MATLAB

```
% sum_to_traced.m
function result = sum_to_traced(n)
    fprintf('Call %d\n', n);
    if n == 0
        result = 0;
    else
        result = n + sum_to_traced(n - 1);
    end
    fprintf('Return %d %d\n', n, result);
end
```

L'affichage "Call" est placé avant le test. L'affichage "Return" précède la sortie et indique la valeur qui sera transmise. Les messages n'interviennent pas dans la formule : ils servent à observer une exécution. Gardez la fonction MATLAB dans `sum_to_traced.m`.

MATLAB

```
% use_sum_to_traced.m
result = sum_to_traced(3);
disp(result); % 6
```

Lire la trace complète

Appels dans un sens, résultats dans l'autre

Descente	Remontée
Call 3	Return 0 : 0
Call 2	Return 1 : 1
Call 1	Return 2 : 3

Descente	Remontée
Call 0	Return 3 : 6

Les deux colonnes ne sont pas exécutées ligne par ligne ensemble : on lit d'abord toute la colonne de gauche, puis celle de droite de haut en bas. Le cas de base ne produit pas la somme finale ; il fournit la première valeur permettant aux appels en attente de reprendre.

Expliquer une valeur en attente

▼ Dans l'appel pour 2, que vaut n après le retour de l'appel pour 1 ?

Il vaut toujours 2. L'appel pour 1 a renvoyé une valeur, sans transformer le n local de son appelant. On peut donc calculer 2 + 1 et renvoyer 3.

▼ Pourquoi voit-on Return 0 avant Return 3 ?

L'appel initial pour 3 dépend de l'appel pour 2, qui dépend de 1, qui dépend de 0. Il doit attendre que ces dépendances se résolvent. C'est le résultat du cas de base qui déclenche la remontée.

Une représentation, pas une nouvelle structure à implémenter

On parle souvent d'une pile d'appels pour décrire cet ordre d'attente et de reprise. Comprendre ce mécanisme ne demande pas d'implémenter une structure pile. Cette étude de structure relève de DSA ; ici, nous suivons simplement l'exécution d'une fonction.

À retenir - fiche mémo

Pile

Les appels commencés mais non terminés restent en attente.

État local

Chaque appel conserve son propre paramètre et son point de reprise.

Retour

Le dernier appel commencé peut terminer avant les appels qui l'attendent.

Pseudo-code

```
// example.pseudo
// pending_calls
// sum_to(3): 3 + sum_to(2)
// sum_to(2): 2 + sum_to(1)
// sum_to(1): 1 + sum_to(0)
// Results: 0, 1, 3, 6.
```

Point de vigilance. Les messages Call et Return sont des observations ajoutées ; ils ne réalisent pas la somme.

Consulter la cheatsheet : Récursivité · **Télécharger la fiche du thème en PDF**

Version web : https://www.clearcode.fr/courses/programming/algorithmic/functions/recursion/tracing-calls