ALGORITHMIQUE · CHAPITRE 30 · MISE EN PRATIQUE

Construire de petits programmes complets

Construisez un calcul de commande, une analyse de mesures et un traitement de commandes. Définissez les règles, tracez les étapes et vérifiez les limites.

Par Ali ZAINOUL · Cours gratuit

Ce que vous allez apprendre

- Préciser un contrat avant d'assembler des instructions.
- Réutiliser conditions, boucles et fonctions.
- Vérifier les frontières et expliquer chaque résultat.

Prérequis. Avoir travaillé les conditions, boucles, fonctions et cas limites.

Construire une solution de bout en bout

Pour chaque projet, commencez par écrire les entrées autorisées, la sortie et deux résultats attendus calculés à la main. Dessinez ensuite le chemin des décisions ou des répétitions, puis écrivez le pseudo-code. Les exemples ci-dessous constituent des solutions de référence à comparer à votre propre méthode, pas des instructions à recopier sans explication.

Trois problèmes, trois contrats explicites

Un **mini-projet** assemble des notions connues autour d'un résultat complet et vérifiable. Nous n'ajoutons ni base de données, ni formulaire réseau, ni système réel de paiement. Chaque entrée est fixée dans le script pour que le calcul puisse être prévu à la main avant l'exécution.

Pour la commande, price est un prix unitaire entier en centimes et quantity une quantité entière non négative. Une commande vide coûte zéro. Sinon, les frais valent 100 centimes si le sous-total est strictement inférieur à 1 000 centimes, et zéro à partir de ce seuil. Le seuil est une constante de règle métier ; le chapitre sur les variables a expliqué comment nommer une telle valeur sans prétendre la verrouiller.

Pour les mesures, la **moyenne arithmétique** est la somme des valeurs divisée par leur nombre. Le **minimum** est la plus petite valeur et le maximum la plus grande. Avec [4, 1, 7], la somme vaut 12, la moyenne 4, le minimum 1 et le maximum 7. Une suite vide n'a ni moyenne ni extrêmes dans ce contrat ; elle déclenche le message `"No measurements"`.

Pour les commandes textuelles, une **commande** est une chaîne reçue dans une suite préétablie. `"add"` augmente le compteur, `"stop"` arrête le parcours et toute autre chaîne affiche `"Unknown command"`. `"help"` est donc volontairement inconnue dans cette version. Cette règle minimale doit être distinguée de ce que son nom pourrait suggérer dans une autre application.

Projet 1 : un calculateur de tarif

Les prix sont des centimes entiers non négatifs ; la quantité est un entier non négatif. Une commande vide coûte zéro. Sinon, la livraison coûte 100 centimes lorsque le sous-total est inférieur à 1000, et elle est offerte à partir de 1000 inclus. Ces règles fictives sont le contrat de cet exercice.

Pseudo-code

```
// example.pseudo
// order_total
FUNCTION order_total(price, quantity)
    IF quantity == 0 THEN
        RETURN 0
    END IF
    subtotal <- price * quantity
    delivery <- 0
    IF subtotal < 1000 THEN
        delivery <- 100
    END IF
    RETURN subtotal + delivery
END FUNCTION
```

Python

```
# order_total.py
def order_total(price, quantity):
    if quantity == 0:
        return 0
    subtotal = price * quantity
    delivery = 0
    if subtotal < 1000:
        delivery = 100
    return subtotal + delivery

total_order = order_total(250, 3)
print(total_order)  # 850
```

MATLAB

```
% order_total.m
function total = order_total(price, quantity)
    if quantity == 0
        total = 0;
        return;
    end
    subtotal = price * quantity;
    delivery = 0;
    if subtotal < 1000
        delivery = 100;
    end
    total = subtotal + delivery;
end
```

En MATLAB, enregistrez la fonction dans `order_total.m`. Le calcul est séparé de l'affichage : on peut vérifier plusieurs commandes sans modifier la fonction.

MATLAB

```
% use_order_total.m
total_order = order_total(250, 3);
disp(total_order); % 850
```

▼ Quels résultats pour (250, 0), (250, 3) et (250, 4) ?

Respectivement 0, 850 et 1000 centimes. Le premier cas teste la commande vide ; le dernier teste exactement la frontière de livraison offerte. Un test strictement supérieur à 1000 ferait payer à tort la livraison du dernier cas.

Projet 2 : analyser une suite de mesures

L'entrée est une suite de nombres réels finis. Pour une suite non vide, produire la somme, la moyenne arithmétique, le minimum et le maximum. Pour une suite vide, annoncer qu'aucune mesure n'est disponible et ne calculer ni moyenne ni extrêmes. La moyenne est la somme divisée par le nombre de valeurs.

Pseudo-code

```
// example.pseudo
// analyze_measurements
values <- [4, 1, 7]
n <- length(values)
IF n == 0 THEN
    PRINT "No measurements"
ELSE
    total <- 0
    minimum <- values[0]
    maximum <- values[0]
    FOR EACH value IN values DO
        total <- total + value
        IF value < minimum THEN
            minimum <- value
        END IF
        IF value > maximum THEN
            maximum <- value
        END IF
    END FOR
    average <- total / n
    PRINT total, average, minimum, maximum
END IF
```

Python

```python
# analyze_measurements.py
values = [4, 1, 7]
n = len(values)
if n == 0:
    print("No measurements")
else:
    total = 0
    minimum = values[0]
    maximum = values[0]
    for value in values:
        total = total + value
        if value < minimum:
            minimum = value
        if value > maximum:
            maximum = value
    average = total / n
    print(total, average, minimum, maximum)  # 12 4.0 1 7
```

MATLAB

```
% analyze_measurements.m
values = [4, 1, 7];
n = numel(values);
if n == 0
    disp('No measurements');
else
    total = 0;
    minimum = values(1);
    maximum = values(1);
    for index = 1:n
        value = values(index);
        total = total + value;
        if value < minimum
            minimum = value;
        end
        if value > maximum
            maximum = value;
        end
    end
    average = total / n;
    disp([total, average, minimum, maximum]); % [12 4 1 7]
end
```

▼ Que doit-on vérifier avec une seule mesure et une suite entièrement négative ?

Avec une seule mesure, somme, moyenne, minimum et maximum valent cette mesure. Avec des mesures négatives, initialiser un maximum à zéro serait incorrect. L'initialisation au premier élément protège ces cas ; le vide doit être traité avant cet accès.

Projet 3 : un menu de commandes avec arrêt

Nous simulons une source de commandes connue : `add` augmente le compteur, `stop` arrête la lecture et toute autre commande produit un message sans augmenter le compteur. Les commandes après stop ne doivent pas être traitées. Cette version utilise une suite finie, sans saisie interactive ni code exécuté depuis du texte.

Pseudo-code

```
// example.pseudo
// process_commands
commands <- ["add", "help", "add", "stop", "add"]
index <- 0
count <- 0
WHILE index < length(commands) DO
    command <- commands[index]
    IF command == "stop" THEN
        BREAK
    END IF
    IF command == "add" THEN
        count <- count + 1
    ELSE
        PRINT "Unknown command"
    END IF
    index <- index + 1
END WHILE
PRINT count
```

Python

```python
# process_commands.py
commands = ["add", "help", "add", "stop", "add"]
index = 0
count = 0
while index < len(commands):
    command = commands[index]
    if command == "stop":
        break
    if command == "add":
        count = count + 1
    else:
        print("Unknown command")
    index = index + 1
print(count)  # 2
```

MATLAB

```matlab
% process_commands.m
commands = {'add', 'help', 'add', 'stop', 'add'};
index = 1;
count = 0;
while index <= numel(commands)
    command = commands{index};
    if strcmp(command, 'stop')
        break;
    end
    if strcmp(command, 'add')
        count = count + 1;
    else
        disp('Unknown command');
    end
    index = index + 1;
end
disp(count); % 2
```

`break` quitte la boucle courante ; il ne termine pas tout le programme. L'affichage final est donc encore effectué. MATLAB conserve les textes dans une cellule : `commands{index}` extrait ici le texte de la commande, et `strcmp` compare les textes. Ces opérations servent à lire notre exemple, sans développer une bibliothèque de structures de données.

▼ Combien d'ajouts pour add, help, add, stop, add ?

Deux. help est signalé mais ne change pas le compteur. stop quitte la boucle avant l'ajout suivant. Si l'on retire stop, les trois commandes add seront traitées, puis la fin de la suite arrêtera la lecture.

Faire relire le programme par ses résultats

- Montrer un cas nominal et expliquer chaque étape.
- Tester une entrée vide ou minimale lorsque le contrat l'autorise.
- Tester exactement les frontières de décision.
- Vérifier que les variables ne sont pas réinitialisées au mauvais endroit.
- Dire pourquoi la boucle ou les appels finissent sur les entrées prévues.

Si une entrée est hors du domaine, choisissez et documentez le comportement avant d'ajouter des contrôles. Les exemples ne constituent pas une application commerciale ou un traitement universel de fichiers : ils rendent des méthodes élémentaires observables.

Ce que vous pouvez désormais construire

Vous pouvez transformer un petit énoncé en données, décisions, répétitions et fonctions, puis confronter le résultat à un contrat. Pour poursuivre, revenez aux chapitres dont une étape reste difficile à expliquer. Les **structures de données et algorithmes** constituent ensuite un approfondissement pour choisir d'autres représentations et méthodes.

À retenir - fiche mémo

Commande

Vide : zéro ; sinon frais de 100 centimes sous un sous-total de 1 000 centimes.

Mesures

Somme, moyenne et extrêmes exigent une règle explicite pour la suite vide.

Commandes textuelles

add compte ; stop arrête ; les autres textes sont inconnus dans cette version.

Pseudo-code

```
// example.pseudo
// project_boundaries
// Empty order: expected 0 cents.
// Measurements [4, 1, 7]: sum 12, mean 4, min 1, max 7.
// Commands ["add", "stop", "add"]: expected count 1.
```

Point de vigilance. Le nom help ne lui donne pas automatiquement un comportement : seule la règle annoncée compte.

Consulter la cheatsheet : Vérifications et projets · **Télécharger la fiche du thème en PDF**

Version web : https://www.clearcode.fr/courses/programming/algorithmic/practice/mini-projects