Formes d'appel – arguments

- Passage **direct explicite**
  - inverse(3)
  - On fournit la **valeur** en cause
  - Justifié quand l'argument en cause est fixe à la ligne donnée du programme
    - p. ex. une fonction *rac* calcule la racine *n*-ième d'un nombre ; il faut calculer $a + \sqrt{a} + \sqrt[3]{a}$, alors on écrit a+rac(a,2)+rac(a,3)
- Passage **direct implicite**
  - inverse(nbre)
  - On fournit le nom d'une **variable** qui contient la valeur en cause
  - En général, l'argument peut être une expression : inverse(2*nbre1+nbre2)
  - Cas standard car la plupart de valeurs utilisées dans des programmes sont des paramètres qu'on garde dans des variables
- Il existe aussi le passage **indirect** (par référence)
  - On fournit la *référence*, c'est-à-dire l'**adresse** de l'emplacement dans la mémoire où est gardée la valeur en cause (p. i. explicite)
    - ou le nom d'une **variable** qui contient cette adresse (p. i. implicite)
  - plus compliqué mais souvent plus efficace ; inexistant (en principe) en Matlab

Formes d'appel – résultats

- Résultat sauvegardé dans une variable
  - inver = inverse(3) ou inver = inverse(nbre)
    - la forme du passage d'arguments est indépendante de celle du renvoi de résultats
  - Cas standard car normalement les calculs sont effectués en plusieurs étapes alors les résultats antérieurs doivent être temporairement mémorisés afin de pouvoir servir d'arguments dans des opérations ultérieures
- Résultat non sauvegardé, utilisé comme argument d'une autre fonction ou opérande d'une opération
  - rapportLongueurs = rac(inverse(3),2)
  - tva = pttc * taux * inverse(1+taux)
  - Quand le résultat est utilisé juste une fois et immédiatement après obtention
- Résultat non sauvegardé, non utilisé, juste affiché
  - inverse(2) ou inverse(nbre)
  - Pratique pour tester une fonction lors de son développement – voir le résultat qu'elle produit pour une valeur d'entrée donnée
- Il existe aussi le renvoi indirect (par référence)

Valeurs par défaut

- La **valeur par défaut** d'un argument d'une fonction, c'est la valeur qui sera supposée si aucune valeur n'est passée pour cet argument
  - function *resultat* = *nom*(*argument_1*, … *argument_i*, *argument_i+1* = *valeur_i+1*, … *argument_n* = *valeur_n*)
  - une même fonction peut posséder des arguments qui ont et qui n'ont pas une valeur par défaut
  - l'ordinateur doit être capable de déterminer quels arguments ont été omis
  - l'ordinateur supposera l'omission à commencer du dernier argument
  - les arguments possédant des valeurs par défaut doivent être placés à la fin
- Exemple
  - function racine = rac(radicande, degre=2)
    racine = radicande^(1/degre);
    endfunction
  - > rac(8,3)
    ans = 2
    > rac(8,2)
    ans = 2.8284
    > rac(8)
    ans = 2.8284
- … tandis que
  - function racine = rac(radicande, degre)
    racine = radicande^(1/degre);
    endfunction
  - > rac(8)
    error : degre undefined

4. Structures de contrôle

Algorithmes simples de traitement de données
Classification des structures de contrôle
Boucle incrémentale
Opérateurs de relation et logiques
Alternatives
Boucles conditionnelles
Instructions de terminaison
Comparaison des boucles

Développement d'algorithmes (exemples)

- Moyenne :
  Calculer la moyenne des éléments d'un vecteur de nombres

- Problème déjà cité :
  Déterminer le numéro du mois où la valeur des ventes a été la plus basse. S'il y a plusieurs mois où la même valeur des ventes minimale est apparue, renvoyer le numéro du premier d'entre eux.
- Simplifions d'abord :
  Trouver la valeur minimale dans un vecteur de nombres.
- Et si on modifie ?
  S'il y a plusieurs mois où la même valeur des ventes minimale est apparue, renvoyer le numéro du dernier d'entre eux.

Structures de contrôle

- Les **structures de contrôle** définissent *l'ordre de l'exécution* des instructions d'un programme
  - Cette ordre peut devenir non linéaire
- La plupart sont trouvées dans tous les langages
- Structures de **répétition** (boucles)
  - pour
  - tant que
  - jusqu'à ce que
- Structures de **condition** (alternatives)
  - si / sinon
  - selon / cas

Structures de blocs

- Instructions de saut
  - saut *
  - sous-programme *
- Instructions de terminaison
  - fin du passage
  - sortie de la boucle
  - sortie de la fonction/procédure
  - arrêt du programme
- Structures de gestion d'exceptions
  - signalement (lance)
  - interception (essaye / attrape)
  - réparation, reprise *

* *inexistantes en Matlab*