Structures de répétition « tant que » (while) et « jusqu'à ce que » (until)

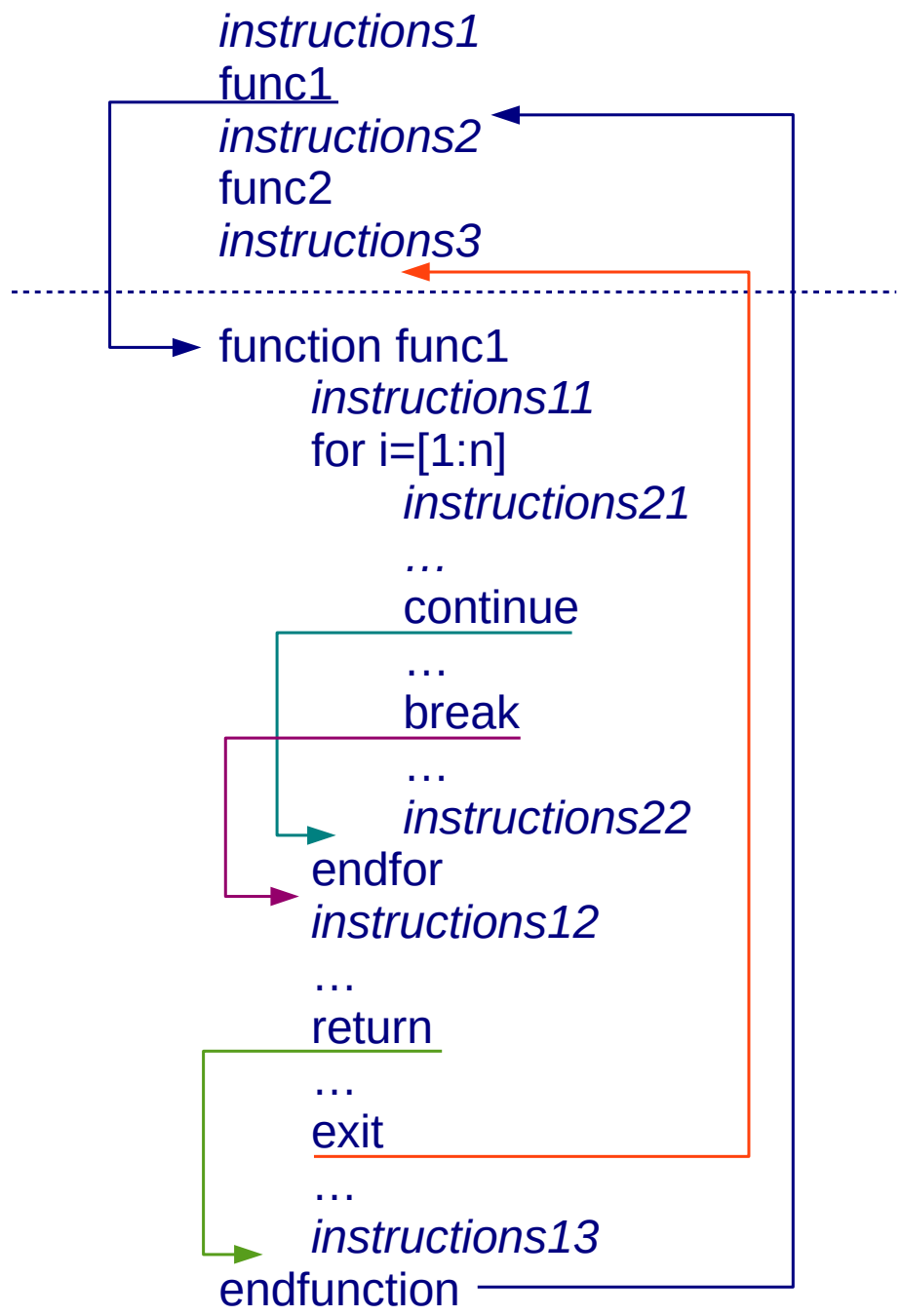
- Décrivent une **boucle conditionnelle**
  - **while** (*condition*)
    instructions
    endwhile
  - **do**
    instructions
    until (*condition*)
- Une telle boucle est répétée :
  - tant que la condition est remplie
    respectivement
  - jusqu'à ce que la condition soit remplie
- Ces formes ne sont pas universelles
  - différentes ou même inexistantes dans d'autres langages
- Il n'y a pas de variable compteur automatique
  - (contrairement au *pour*)
  - On peut l'introduire et modifier sa valeur explicitement mais ça va produire un code plus long
- Le nombre de répétitions n'est pas défini au début
  - (ce qui est normalement le cas avec *pour*)
  - Plutôt que répéter un nombre de fois défini, la fin de la boucle est déterminée par vérification d'une condition
  - Si on introduit une variable compteur, la condition la peut contenir

Instructions de terminaison

- continue – fin du passage
  - Le passage en cours de la boucle est fini et le prochain passage est commencé
  - ```
    while (n <= nmax)
        if (vect(n) <= 0)
            continue
        endif
        instructions
    endwhile
    ```
  - Utilité dans ce cas :
    si les instruction*s* sont nombreuses et on sait qu'elles ont un sens pour des valeurs positives seulement
- break – sortie de la boucle
  - Le passage en cours est fini et aucun autre passage n'est effectué
  - ```
    for n = [1:nmax]
        if (vect(n) < 0)
            break
        endif
        instructions
    endfor
    ```
  - Utilité dans ce cas :
    si le calcul doit être terminé dès que les valeurs deviennent négatives

Instructions de terminaison (suite)

- return – sortie de la fonction
  - Les instructions suivantes de la fonction ne sont pas exécutées
  - Le résultat est toujours renvoyé
    - Il est alors important qu'il soit défini (variable résultat affectée) dans toutes les circonstances possibles (celles qui provoquent les *continue*, *break* ou *return*)
- exit – arrêt du programme
  - L'exécution du programme entier est momentanément terminée
  - y compris toutes fonctions appelées depuis ce programme



Structures de répétition – exemple comparatif

- Trouver le dernier zéro dans un vecteur et renvoyer sa position ou 0 si non trouvé
  - **indentation** :

    1er niveau – les instructions qui appartiennent à la fonction

    2e niveau – d'entre les précédentes, celles qui sont répétées en boucle

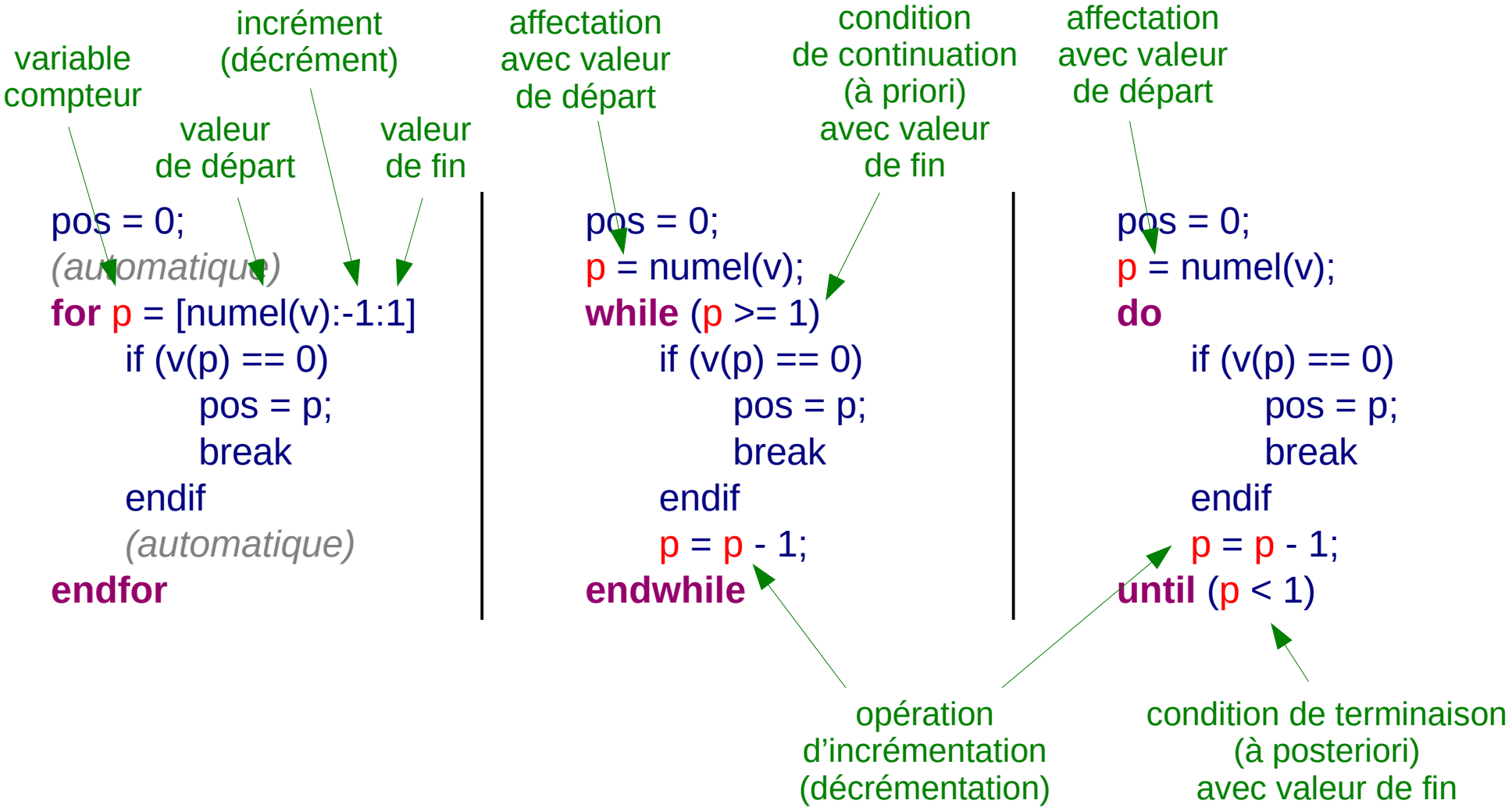
    3e niveau – d'entre les précédentes, celles qui sont exécutées sous une condition

```
function pos=trouverzero(v)
    pos = 0;

    for p = [numel(v):-1:1]
        if (v(p) == 0)
            pos = p;
            break
        endif

    endfor
endfunction
```

```
function pos=trouverzero(v)
    pos = 0;
    p = numel(v);
    while (p >= 1)
        if (v(p) == 0)
            pos = p;
            break
        endif
        p = p - 1;
    endwhile
endfunction
```

```
function pos=trouverzero(v)
    pos = 0;
    p = numel(v);
    do
        if (v(p) == 0)
            pos = p;
            break
        endif
        p = p - 1;
    until (p < 1)
endfunction
```

Structures de répétition – exemple comparatif (suite)



« itérer de numel(v) à 1 avec l'incrément de −1 »

« répéter tant que *p* reste supérieur ou égal 1 »

« répéter jusqu'à ce que *p* devient inférieur à 1 »

Versions d'un vrai programmeur

```
pos = 0;

for p = [numel(v):-1:1]
      if (v(p) == 0)
            pos = p;
            break
      endif

endfor
```

```
pos = 0;
p = numel(v);
while (p >= 1)
      if (v(p) == 0)
            pos = p;
            break
      endif
      p = p - 1;
endwhile
```

```
pos = 0;
p = numel(v);
do
      if (v(p) == 0)
            pos = p;
            break
      endif
      p = p - 1;
until (p < 1)
```

1. On tente d'éviter ≤, ≥ ainsi que de comparer à 0 si possible (ça augmente la vitesse d'exécution bas-niveau)
2. Avec *while* et do-*until* on peut combiner la variable compteur et la variable résultat en une seule.

```
pos = 0;
for p = [numel(v):-1:1]
      if (v(p) == 0)
            pos = p;
            break
      endif
endfor
```

```
pos = numel(v);
while (pos > 0)
      if (v(pos) == 0)
            break
      endif
      pos = pos - 1;
endwhile
```

```
pos = numel(v);
do
      if (v(pos) == 0)
            break
      endif
      pos = pos - 1;
until (pos == 0)
```

Critères du choix d'une structure de répétition

- Le choix entre *for*, *while* et *do* dépendra de l'algorithme que l'on tente à mettre en œuvre
- On choisira la structure qui permettra d'obtenir un code optimal :
  - moins d'actions (à l'intérieur de la boucle mais aussi avant et après)
  - moins de variables (économie de la mémoire)
  - condition moins compliquée
  - possibilité d'éviter les instructions de terminaison
- L'exemple analysé :
  - nombre de lignes du code identique 😐
  - une ligne additionnelle à l'intérieur de la boucle (décrémentation de *pos*) avec *while* et *do* ; mais cette action est aussi cachée dans *for* 😐
  - une ligne additionnelle avant la boucle (affectation *pos*) – cachée dans *for* 😐
  - effectivement 2 actions de plus avec *for* (pos = 0, pos = p) ; pourtant ça ne change pas considérablement le temps d'exécution (exécutées juste 1 fois) ☹
  - 1 variable de plus avec *for* (*p*) ; peu important (juste 1 cellule de mémoire) ☹
  - conditions de la même complexité ; *break* utilisé dans tous les cas 😐
  - structure et mise en œuvre du compteur plus simples avec *for* 🙂🙂

5. Représentation et stockage des données

Représentation des données par l'ordinateur
Formatage style C de sortie et d'entrée
Fichiers, leurs types et gestion
Écriture et lecture simples et avec formatage

Représentation des données par l'ordinateur

- Forme « naturelle » décimale (base 10) : 158
  - En fait, toute donnée est binaire dans la mémoire
  - Forme binaire (base 2) : 1001 1110
  - Le même en notation hexadécimale (base 16) : 9E
    - A B C D E F désignent : 10 11 12 13 14 15
- **8 bits** nécessaires = **1 octet**
- Bit et octet (byte) sont les **unités de quantité de données**
  - bit : 1 b ≡ 1 chiffre binaire
  - octet : 1 B (1 o) = 8 b
- Forme « naturelle » textuelle (chaîne de caractères) : "158"
  - Décomposition en vecteur de caractères ["1", "5", "8"]
  - Codage ASCII (0...255) [49, 53, 56]
  - Forme binaire [00110001, 00110101, 00111000]
  - Notation hexadécimale [31, 35, 38]
    - pour le discerner des nombres 31, 35 et 38 décimaux on ajoute : 31h 35h 38h ou 0x31 0x35 0x38
- **24 bits** nécessaires = **3 octets**

Forme des données dans un programme Matlab

- C'est le programme (donc le programmeur) qui détermine si une donnée est considérée et mémorisée :
  - comme un nombre
  - ou comme une chaîne de caractères
- Des exemples que l'on a déjà vues
  - variable1 = 527
    variable2 = "527"
  - input(entree)
    input(entree, "s")
- Plus compliqué (mais aussi fiable) dans d'autres langages
  - Il y faut déclarer explicitement le type de chaque variable
- Une que l'on va voir de suite
  - printf("%d", variable1)
    printf("%s", variable2)
- Ne soyez donc pas surpris :
  - > a = 3.5
    a = 3.5000
    > a + 1
    ans = 4.5000
  - > b = "3.5"
    b = 3.5
    > b + 1
    ans = 52 47 54
  - la valeur de la variable « b » a été saisie en forme chaîne de caractères
  - donc un vecteur de codes des caractères "3", "." et "5"
  - donc [51 46 53]
  - si on additionne 1 à ce vecteur...