Structure de condition « si / sinon » (if / else)

- Structure simple
  - **if** (*condition*)
    instructions
    endif
  - Si la *condition* est remplie, alors les *actions* sont exécutées
  - Sinon, aucune action n'est prise
- La *condition*
  - une **expression logique** quelconque, c.-à-d. une expression à laquelle peut être attribuée une valeur logique (vrai / faux)
- Structure avec alternative
  - if (*condition*)
    instructions
    else
    autres_instructions
    endif
  - Si la *condition* est remplie, alors les *instructions* sont exécutées
  - Sinon, les *autres_instructions* sont exécutées

Structure de condition « si / sinon » (suite)

- Structure avec alternatives multiples
  - if (*condition_1*)
    instructions_1
    elseif (*condition_2*)
    instructions_2
    elseif (*condition_3*)
    instructions_3
    …
    elseif (*condition_n*)
    instructions_n
    else
    instructions_nn
    endif
- La partie « else » n'est pas obligatoire
  - Il peut n'y avoir aucune alternative finale
  - si *condition_1* est remplie (*true* ou 1)
    - les *instructions_1* sont exécutées
    - aucune autre condition n'est vérifiée
  - sinon (si *condition_1* n'est pas remplie alors sa valeur est *false* ou 0)
    - la *condition_2* est vérifiée
    - si *condtion_2* est remplie
      - *instructions_2* sont exécutées
      - aucune autre condition n'est vérifiée
    - sinon (*condition_2* n'est pas remplie)
      - *condition_3* est vérifiée
      - …
  - si aucune des *conditions* 1 à *n* n'est remplie
    - *instructions_nn* sont exécutées

Structure « si / sinon » – exemples

- Structure avec alternative :
  - Mémoriser le supérieur de deux nombres dans une troisième variable
  - ```
    if (a > b)
        max = a;
    else
        max = b;
    endif
    ```
  - Ça marche aussi quand les deux nombres sont égaux (dans ce cas il n'est pas important la valeur de quelle variable est copiée parce que c'est la même valeur)
  - **Indentation** : l'ensemble des instructions qui sont exécutées sous une condition sont décalées vers la droite par rapport à la ligne où cette condition est définie
- Alternatives multiples :
  - Déterminer le signe d'un nombre (1 ou –1) ; 0 s'il est 0
  - ```
    if (n > 0)
        sgn = 1;
    elseif (n < 0)
        sgn = -1;
    else
        sgn = 0;
    endif
    ```
  - La condition « n == 0 » n'apparaît pas parce qu'elle est déduite des deux premières conditions (si aucune n'est remplie, alors *n* doit être égal 0)

Structures imbriquées

- Exemple :
  - Calculer la somme des éléments positifs seulement d'un vecteur
  - ```
    somme = 0;
    for no_elem = [1:numel(v)]
        if v(no_elem) > 0
            somme = ...
            somme+v(no_elem);
        endif
    endfor
    ```
  - p. ex.
    ```
    v = [1 -2 4 -8 16 -32]
    ```
- La structure niveau plus bas (*if*) doit être fermée (*endif*) avant que la structure niveau plus haut (*for*) soit fermée (*endfor*)

- Structures du même type :
  - Calculer la somme des éléments d'une matrice
  - ```
    somme = 0;
    for ligne = [1:rows(M)]
        for colonne = [1:columns(M)]
            somme = ...
            somme+M(ligne,colonne);
        endfor
    endfor
    ```
  - p. ex.
    ```
    M = [1 -2; 4 -8; 16 -32]
    ```
- Il sera assumé que la première structure fermée est la dernière ouverte donc le premier *endfor* concerne le *for colonne* et non pas le *for ligne*

Structure de condition « selon / cas » (switch / case)

- Cas générique :
  - **switch** (*variable*)
    case *valeur_1*
    instructions_1
    case *valeur_2*
    instructions_2
    …
    case *valeur_n*
    instructions_n
    otherwise
    instructions_nn
    endswitch
- Remplacement pour « if » afin de ne pas répéter « elseif »
- Il doit y avoir au moins un *case*
- La partie *otherwise* n'est pas obligatoire

- Attention avec d'autres langages basés sur le C
  - À la fin d'un cas (*case*) il faut mettre *break*
  - Sinon, l'exécution continuera avec les *instructions* du cas suivant
    - Ce n'est pas désiré dans la plupart des cas (pas tous les cas quand même)
- Cas combinés
  - case {194 233}
    banque="Credit Agricole"
  - est équivalent à :
    case 194
    banque="Credit Agricole"
    case 233
    banque="Credit Agricole"

Structure de condition « selon / cas » – exemple

- À comparer :
  - Afficher le descriptif d'une note d'un employé exprimée comme nombre
  - ```
    switch (note)
        case 5
            disp("excellent");
        case 4
            disp("bien");
        case 3
            disp("passable");
        case 2
            disp("insuffisant");
        otherwise
            disp("nombre incorrect");
    endswitch
    ```
  - Plus lisible et plus court
  - Plus facile de changer du nom de la variable ou ajouter plus d'options
- ...avec une solution basée sur *if* (le résultat est identique) :
  - ```
    if (note == 5)
        disp("excellent");
    elseif (note == 4)
        disp("bien");
    elseif (note == 3)
        disp("passable");
    elseif (note == 2)
        disp("insuffisant");
    else
        disp("nombre incorrect");
    endif
    ```
- Afin de pouvoir utiliser *switch* il est nécessaire que chaque condition
  - soit basée sur l'égalité (==)
  - concerne la même variable (ici, *note*)

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

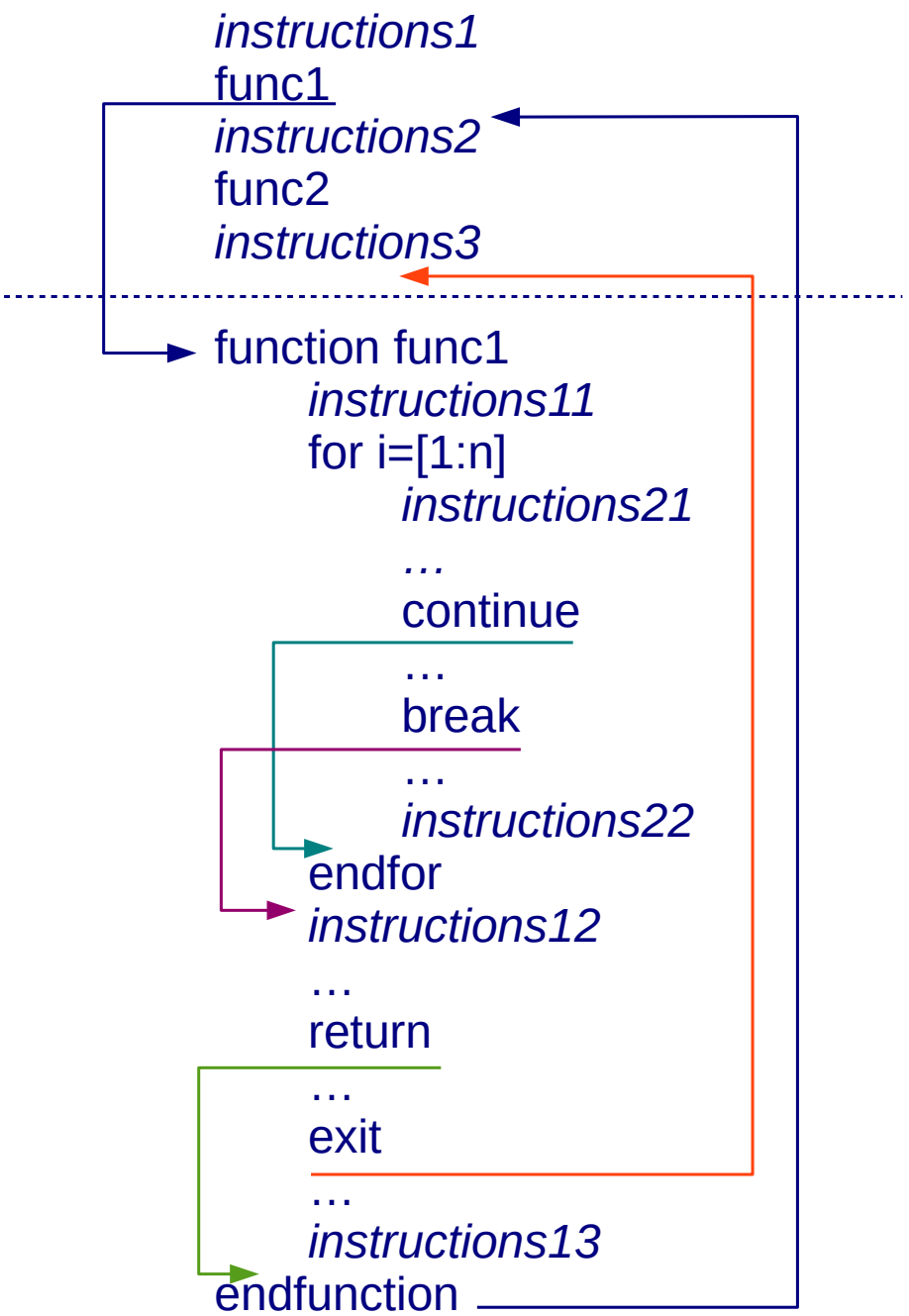
- Décrivent une **boucle conditionnelle**
  - **while** (*condition*)
    instructions
    endwhile
  - **do**
    instructions
    until (*condition*)
- Une telle boucle est répétée :
  - tant que la condition est remplie
    respectivement
  - jusqu'à ce que la condition soit remplie
- Ces formes ne sont pas universelles
  - différentes ou 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 comparative

- Trouver un 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

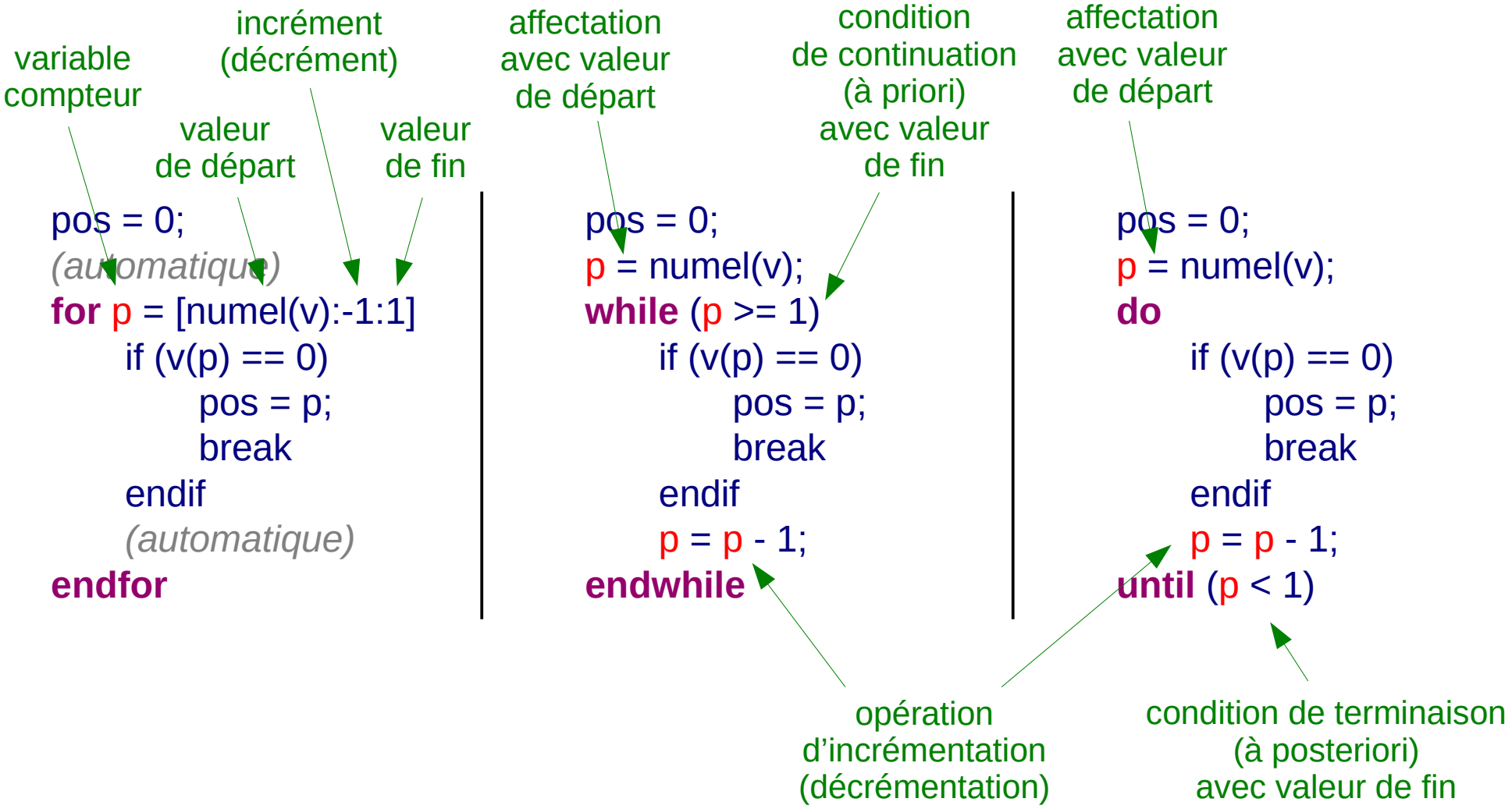
    3e niveau – d'entre les précédentes, celles qui sont exécutées sous une condition

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

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

    endfor
endfunction
```

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

```
function pos=trouver_zero(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 comparative (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
  - pas de nécessité de recourir aux instructions additionnelles (*break* etc.)
- L'exemple analysée :
  - nombre de lignes du code identique =
  - une ligne additionnelle dans 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*) –
  - conditions de la même complexité ; *break* utilisé dans tous les cas =
  - structure et mise en œuvre du compteur plus simples avec *for* +