7 - Paramétrisation des algorithmes : les Actions nommées

Description de modifications d'états

Jean-Michel Adam - UGA - UFR SHS

Deux types de sous-algorithmes

- Les fonctions : noms associés à des **calculs**
- Les actions nommées (ou procédures) : noms associés à des algorithmes qui effectuent une modification de l'état du système
- Etat du système : valeurs des variables mises en jeu par l'action
- Intérêt des actions nommées
  - Permet de décomposer un problème en sous-problèmes plus simples
  - Permet de résoudre un problème en faisant abstraction des sous-problèmes qui seront résolus séparément

Action nommée

- Une action est caractérisée par
  - Son rôle, son effet
  - Son état initial (e.i.)
  - Son état final (e.f.)
- Certaines actions agissant sur des périphériques sont prédéfinies dans la notation algorithmique
  - cl.saisir(liste des arguments) si cl est un clavier
  - e.afficher(liste d'arguments) si e est un écran

Forme générale de réalisation d'une action

En-tête

Action nomaction (liste paramètres formels)

// Effet : description du rôle de l'action

// Etat initial : description de l'état des indicateurs avant l'exécution de l'action

// Etat final : description de l'état des indicateurs après l'exécution de l'action

Corps

lexique de nomaction

définition
- des paramètres de l'action
- des variables locales nécessaires pour décrire l'algorithme

algorithme de nomaction

algorithme permettant de passer de l'état initial à l'état final

Nature des paramètres formels d'une action

- Paramètre **consulté** par l'action : la valeur du paramètre est **utilisée** par l'action
- Paramètre **élaboré** par l'action : la valeur du paramètre est indéterminée à l'état initial, l'action a pour objet de calculer la valeur du paramètre
- Paramètre **modifié** par l'action : la valeur du paramètre peut être modifiée par l'action

Forme générale de réalisation d'une action

En-tête

Action nomaction (consulté x : type, élaboré y : type, modifié z : type)

// Effet : description du rôle de l'action

// Etat initial : $x = x_0$, $y = ?$, $z = z_0$

// Etat final : $x = x_0$, $y = y_1$, $z = z_1$

Corps

lexique de nomaction

définition

- des paramètres de l'action
- des variables locales nécessaires pour décrire l'algorithme

algorithme de nomaction

algorithme permettant de passer de l'état initial à l'état final

Exemples : les actions saisir et afficher

cl.saisir(a,b,c)

// état initial : a = ?, b = ?, c = ?

// état final : a = a_0, b = b_0, c = c_0

Ici nous sommes en présence de paramètres élaborés : leur valeur n'est pas définie au début de l'action, ils sont élaborés par l'action de saisie

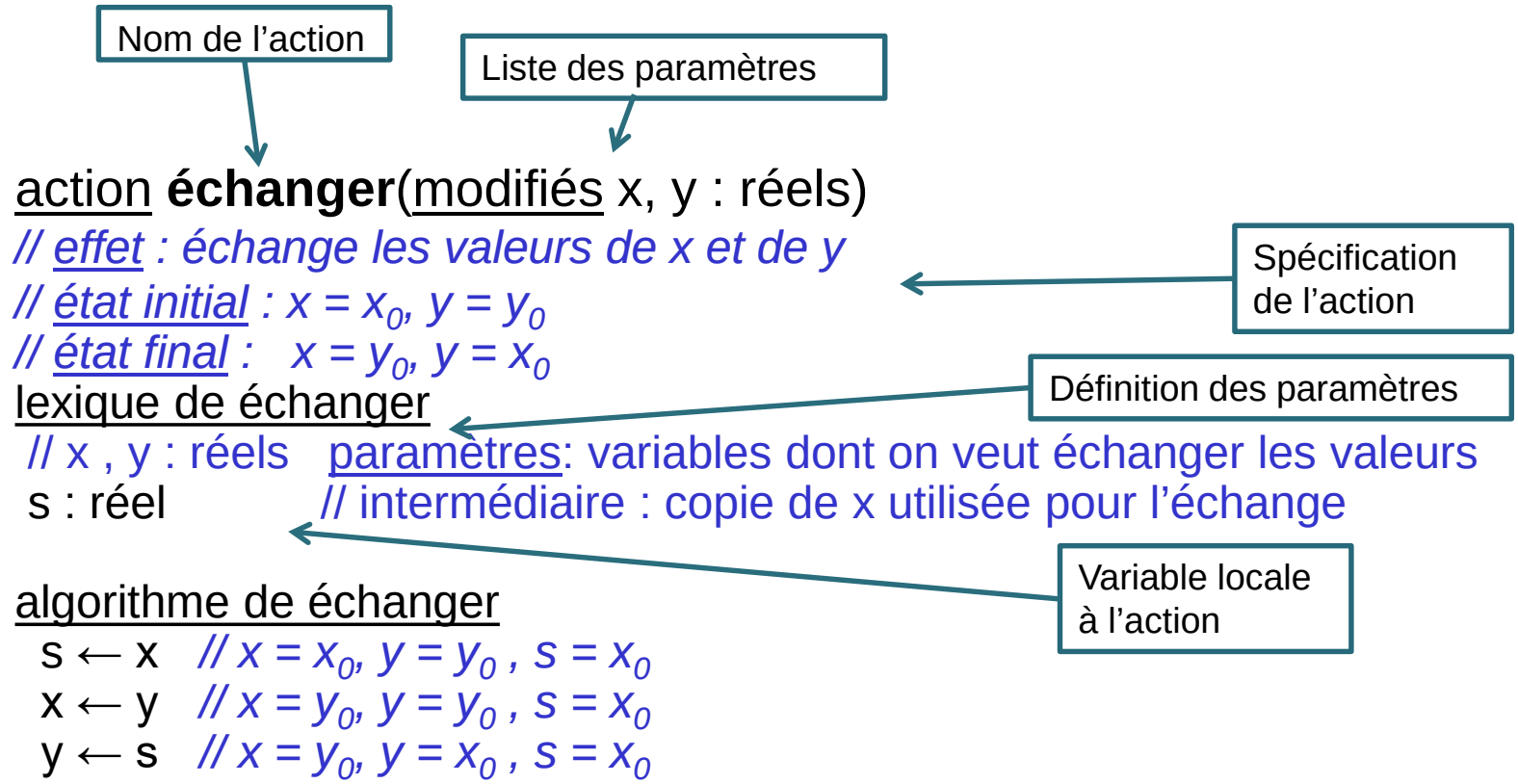
e.afficher(a,b,c)

// état initial: a = a_0, b = b_0, c = c_0

// état final : a = a_0, b = b_0, c = c_0

Ici nous sommes en présence de paramètres consultés : leur valeur n'est pas modifiée par l'action d'affichage, l'état du système modifié ici est l'écran

Réalisation informatique d'une action qui échange les valeurs de ses deux paramètres



Utilisation d'une action

- Une action s'utilise partout où l'on peut utiliser une instruction
- On parle **d'appel** de l'action
- Forme de l'appel:

  nomaction(liste des arguments)
- Un appel de l'action a pour effet d'agir sur l'état du système, selon les arguments donnés dans l'appel

Exemple d'utilisation d'une action : classement de 3 entiers donnés

lexique principal
  cl : clavier // périphérique d'entrée
  e : écran // périphérique de sortie
  a, b, c : entiers // données : valeurs à examiner
  p, s, t : entier // résultats : valeurs ordonnées de a, b, c, de sorte que $p \leq s \leq t$
  action utilisée : échanger

Algorithme principal
  e.afficher("entrez 3 entiers distincts :")
  cl.saisir (a, b, c) // $a = a_0$, $b = b_0$, $c = c_0$
  p ← a ; s ← b ; t ← c // $p = a_0$, $s = b_0$, $t = c_0$
  si p > s
  alors échanger(p,s) // échanger les valeurs de p et de s car p > s
  fsi // $p \leq s$, $s = \max(a_0 , b_0)$
  si s > t
  alors échanger(s,t) // échanger les valeurs de s et de t car s > t
  fsi // $s \leq t$, $t = \max(a_0 , b_0 , c_0)$
  si p > s
  alors échanger(p,s) // échanger les valeurs de p et de s car p > s
  fsi
  // $p \leq s$ et $s \leq t$
  e.afficher(p, ' ', s, ' ', t)

Transformation de l'algorithme précédent en une action de classement de 3 valeurs

action **classer3V**(consultés **x**, **y**, **z** : entiers, élaborés **min**, **int**, **max** : entiers)
// effet : classe les valeurs de x, y et z dans min, int et max de sorte que min ≤ int ≤ max
// e.i. : $x = x_0$, $y = y_0$, $z = z_0$, min = ?, int = ?, max = ?
// e.f. : $x = x_0$, $y = y_0$, $z = z_0$,
// min = minimum(x_0,y_0,z_0), int = intermédiaire(x_0,y_0,z_0), max = maximum(x_0,y_0,z_0)

lexique de classer3V
// x, y, z : entiers paramètres : valeurs à classer
// min,int,max: entier paramètres: valeurs ordonnées de x, y, z, de sorte que min ≤ int ≤ max
action utilisée : échanger

Algorithme de classer3V
// $x = x_0$, $y = y_0$, $z = z_0$
min ← x ; int ← y ; max ← z // $min = x_0$, $int = y_0$, $max = z_0$
si min > int alors échanger(min,int) // échanger les valeurs de min et de int car min > int
fsi // min ≤ int, min = minimum(x_0,y_0), int = maximum(x_0,y_0)
si int > max alors échanger(int,max) // échanger les valeurs de int et de max car int>max
fsi // int ≤ max, min = minimum(x_0,y_0), max = maximum(x_0,y_0,z_0)
// int = minimum(maximum(x_0,y_0), z_0)
si min > int alors échanger(min,int) // échanger les valeurs de min et de int car min > int
fsi
// min ≤ int ≤ max,
// min = minimum(x_0,y_0,z_0), int = intermédiaire(x_0,y_0,z_0), max = maximum(x_0,y_0,z_0)

Exemple d'usage de l'action classer3V

lexique principal
  cl : clavier // périphérique d'entrée
  e : écran // périphérique de sortie
  a, b, c : entiers // données : valeurs à examiner
  p, s, t : entier // résultats : valeurs ordonnées de a, b, c, de sorte que p≤s≤ t
  action utilisée : classer3V

Algorithme principal
  e.afficher("entrez 3 entiers distincts :")
  cl.saisir (a, b, c) // $a = a_0$, $b = b_0$, $c = c_0$
  classer3V(a, b, c, p, s, t)
  // p = minimum(a_0,b_0,c_0), s = intermédiaire(a_0,b_0,c_0), t = maximum(a_0,b_0,c_0)
  // p ≤ s ≤ t
  e.afficher(p, ' ', s, ' ', t)

Transformation de l'algorithme précédent en une action de tri de 3 valeurs

action **trier3V**(modifiés **x**, **y**, **z** : entiers)
// effet : classe les valeurs de x, y et z de sorte que $x \leq y \leq z$
// e.i. : $x = x_0$, $y = y_0$, $z = z_0$
// e.f. : $x = \text{minimum}(x_0,y_0,z_0)$, $y = \text{intermédiaire}(x_0,y_0,z_0)$, $z = \text{maximum}(x_0,y_0,z_0)$
lexique de classer3V
// x, y, z : entiers paramètres : valeurs à classer
action utilisée : échanger

Algorithme de classer3V
// $x = x_0$, $y = y_0$, $z = z_0$
si x > y alors échanger(x,y) // échanger les valeurs de x et de y car x > y
fsi // $x \leq y$, $x = \text{minimum}(x_0,y_0)$, $y = \text{maximum}(x_0,y_0)$
si y > z alors échanger(y,z) // échanger les valeurs de int et de max car y > z
fsi // $y \leq z$, $x = \text{minimum}(x_0,y_0)$, $z = \text{maximum}(x_0,y_0,z_0)$
// $y = \text{minimum}(\text{maximum}(x_0,y_0), z_0)$
si x > y alors échanger(x,y) // échanger les valeurs de x et de y car x > y
fsi
// $x \leq y \leq z$,
// $x = \text{minimum}(x_0,y_0,z_0)$, $y = \text{intermédiaire}(x_0,y_0,z_0)$, $z = \text{maximum}(x_0,y_0,z_0)$

Exemple d'usage de l'action trier3V

lexique principal
  cl : clavier // périphérique d'entrée
  e : écran // périphérique de sortie
  a, b, c : entiers // données : valeurs à examiner
  p, s, t : entier // résultats : valeurs ordonnées de a, b, c, de sorte que p≤s≤ t
  action utilisée : trier3V

Algorithme principal
  e.afficher("entrez 3 entiers distincts :")
  cl.saisir (a, b, c) // $a = a_0$, $b = b_0$, $c = c_0$
  p ← a ; s ← b ; t ← c
  trier3v(p,s,t)
  // p = minimum(a_0,b_0,c_0), s = intermédiaire(a_0,b_0,c_0), t = maximum(a_0,b_0,c_0)
  // p ≤ s ≤ t
  e.afficher(p, ' ', s, ' ', t)

Paramètres et Arguments

- Pour chaque appel d'une action, le nombre **d'arguments** doit correspondre au nombre de **paramètres** de l'action.
- Le **type** du nième argument doit correspondre au type du nième paramètre.
- Les arguments consultés peuvent être des expressions
- Les arguments élaborés et modifiés sont toujours des noms de variables
- Si le nième argument est un nom, il aura, de préférence, un nom différent de celui du nième paramètre.

Fonctionnement de l'appel d'une action

Au moment de l'appel l'action :

- les valeurs des arguments consultés sont affectés aux paramètres correspondants
  - 1er paramètre consulté ← 1er argument correspondant
  - 2ème paramètre consulté ← 2ème argument correspondant
  - etc.
- Les adresses des arguments élaborés ou modifiés sont affectés aux paramètres correspondants. L'action accède aux valeurs des arguments via les paramètres.

Représentation en mémoire des variables et des paramètres : principes généraux

- A l'exécution d'un programme, la mémoire est divisée en deux parties :
  - La partie statique qui contient les variables dont la durée de vie est celle du programme
  - La partie dynamique qui contient les variables et paramètres créés et détruits dynamiquement durant l'exécution du programme.
    - Les créations se font au moment de l'appel de l'action ou de la fonction
    - Les destructions se font à la fin de l'exécution de l'action ou de la fonction

Principes généraux

- Lors de l'appel d'une action ou d'une fonction, les paramètres et les variables locales sont créées
- Les paramètres sont initialisés de la façon suivante :
  - Les paramètres consultés sont initialisés avec la valeur de l'argument. On parle de passage de paramètre par valeur.
  - Les paramètres élaborés ou modifiés sont initialisés avec l'adresse de l'argument, l'accès à la valeur se fait par un adressage indirect. On parle de passage de paramètre par référence.
  - Si un paramètre effectif est une référence, il est transmis par valeur dans tous les cas car c'est une adresse (on transmet la valeur de l'adresse)

Exemple de représentation en mémoire des informations manipulées

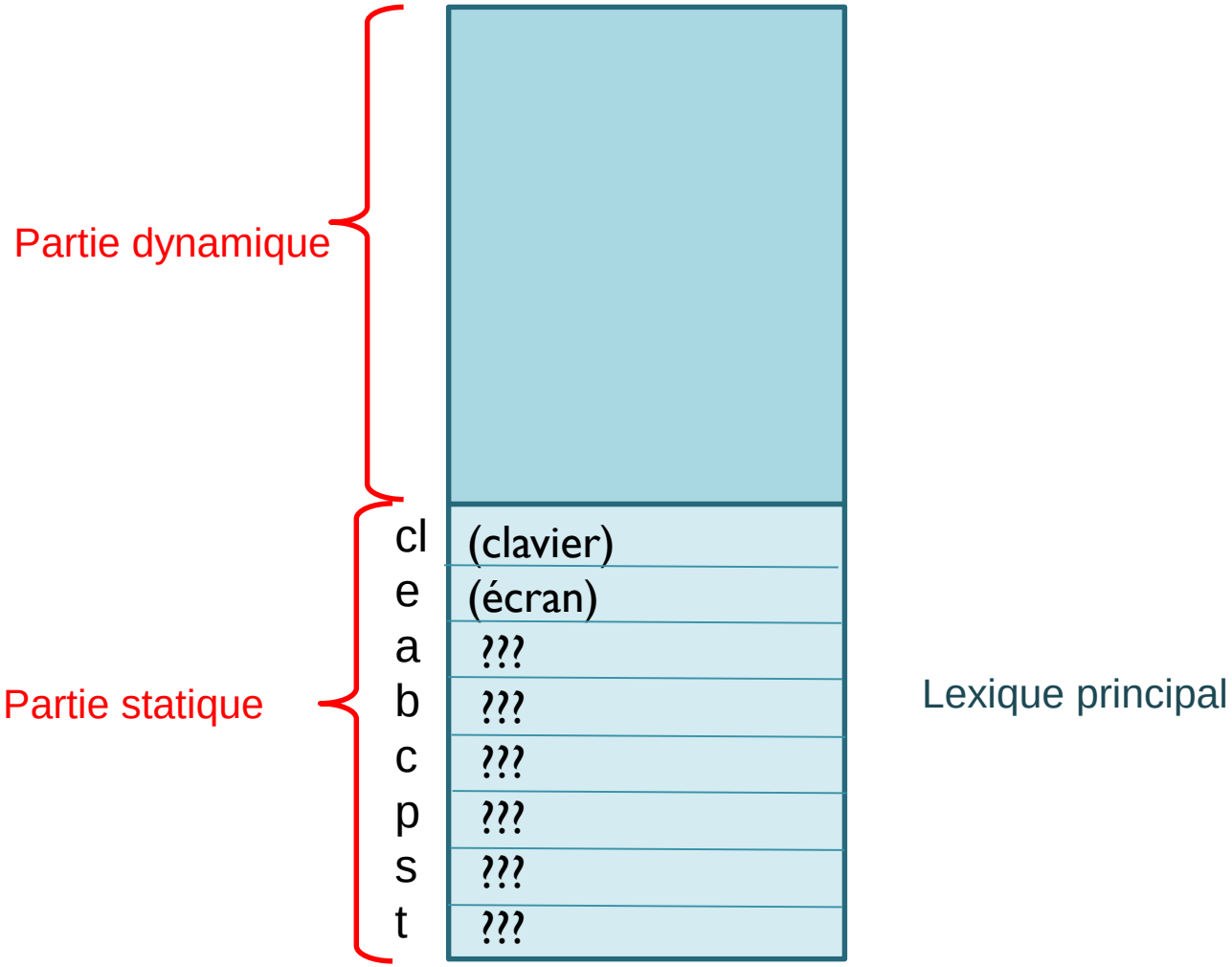
lexique principal
  cl : clavier // périphérique d'entrée
  e : écran // périphérique de sortie
  a, b, c : entiers // données : valeurs à examiner
  p, s, t : entier // résultats : valeurs ordonnées de a, b, c, de sorte que p≤s≤ t
  action utilisée : classer3V

Algorithme principal
  e.afficher("entrez 3 entiers distincts :")
  cl.saisir (a, b, c) // $a = a_0$, $b = b_0$, $c = c_0$
  classer3V(a, b, c, p, s, t)
  // p = minimum(a_0,b_0,c_0), s = intermédiaire(a_0,b_0,c_0), t = maximum(a_0,b_0,c_0)
  // p ≤ s ≤ t
  e.afficher(p, ' ', s, ' ', t)

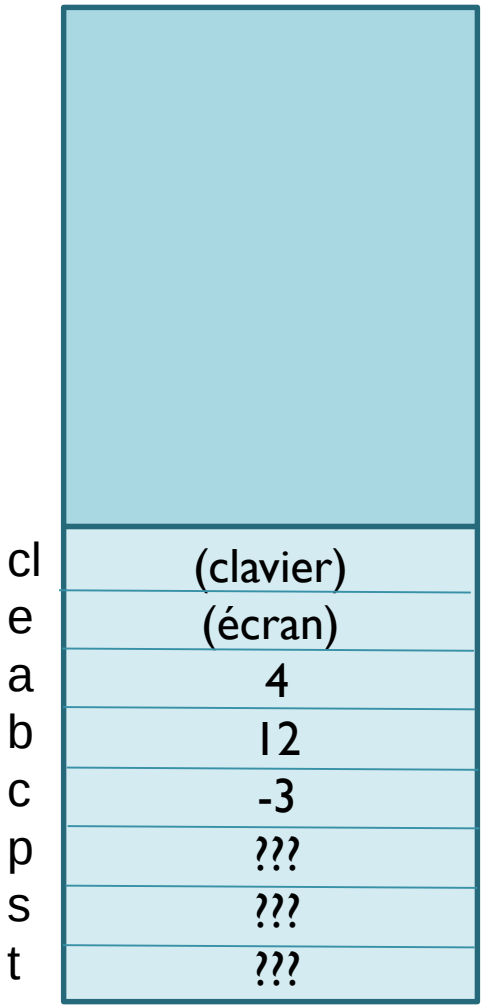
Tous les objets définis dans le lexique principal ou dans le lexique partagé sont définis dans la partie statique.

Dans l'exemple ci-dessus, c'est le cas de cl, e, a, b, c, p, s et t

Etat de la mémoire au début de l'exécution de l'algorithme principal



Saisie des données



Instructions exécutées :

```
e.afficher("entrez 3 entiers distincts :")
cl.saisir (a, b, c)
classer3v(a, b, c, p, s, t)
```

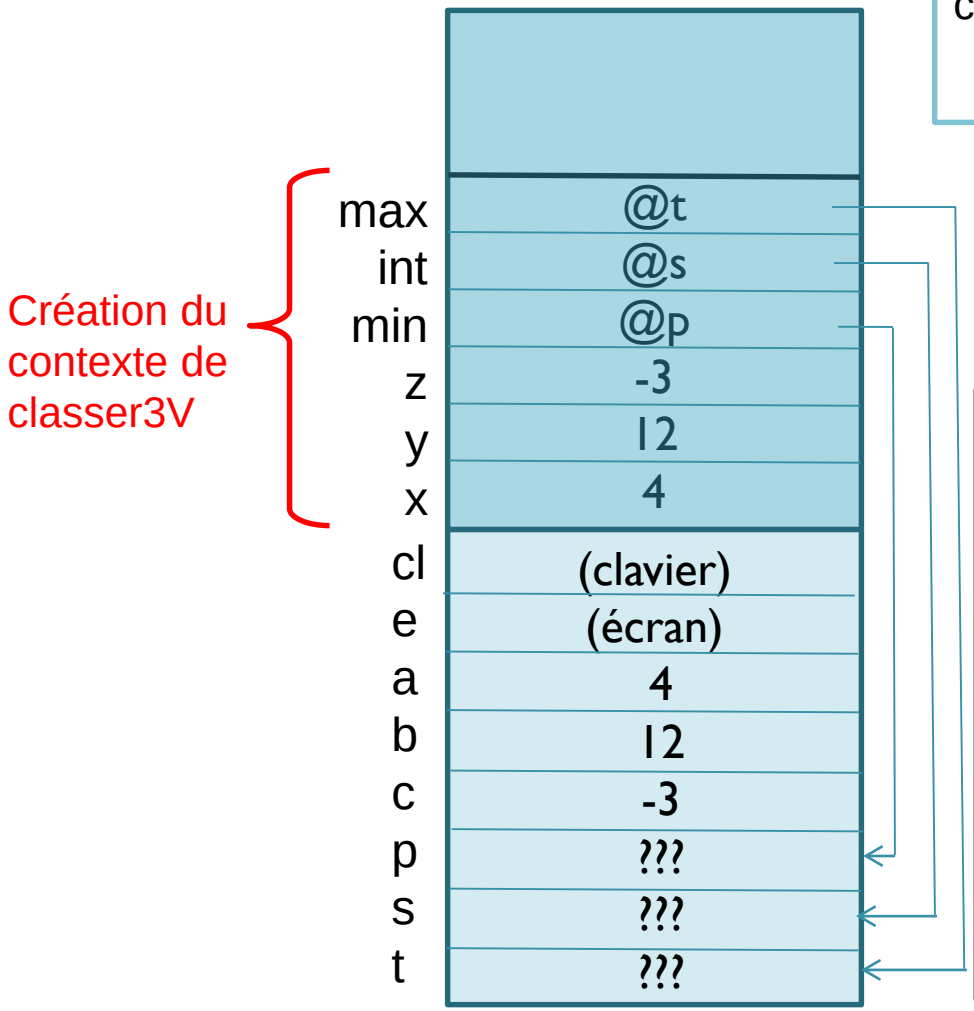
Texte affiché à l'écran:

```
entrez 3 entiers distincts :
4, 12, -3
```

Appel de l'action classer3V

Instructions exécutées :

classer3V(a, b, c, p, s, t)



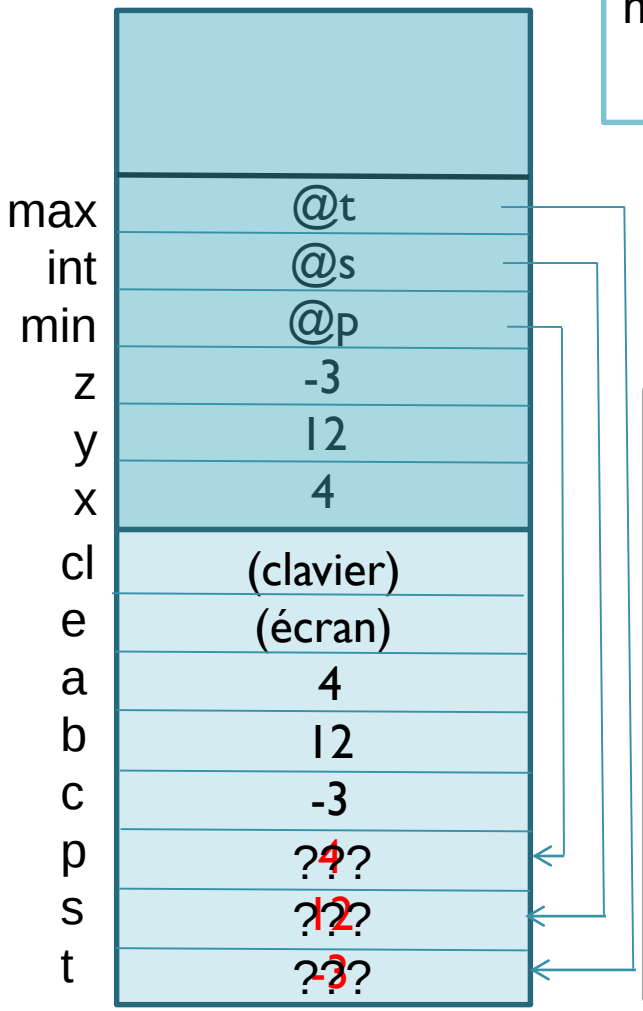
Texte affiché à l'écran:

entrez 3 entiers distincts :
4, 12, -3

Appel de l'action classer3V

Instructions exécutées :

min ← x ; int ← y ; max ← z



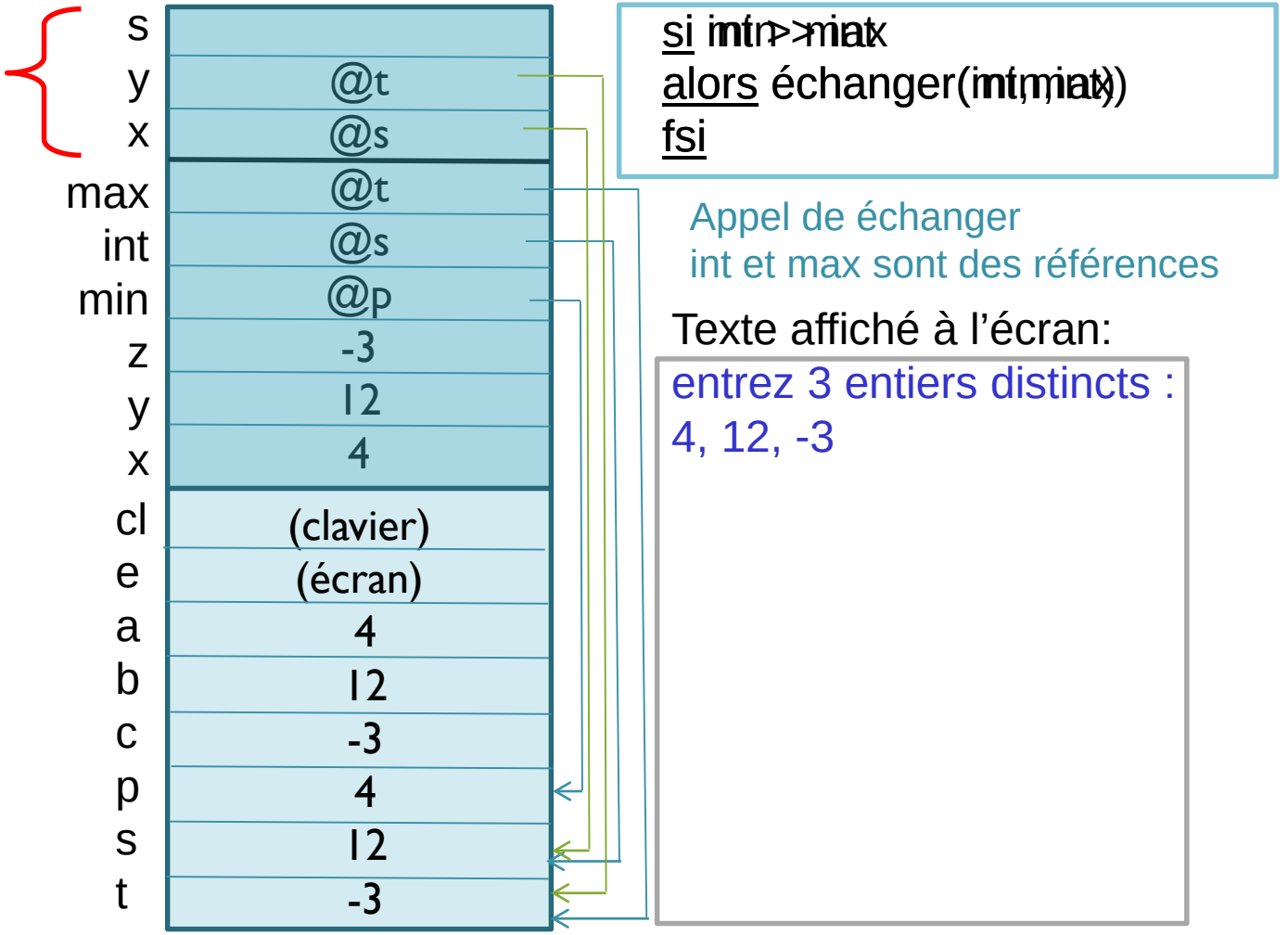
Texte affiché à l'écran:

entrez 3 entiers distincts :
4, 12, -3

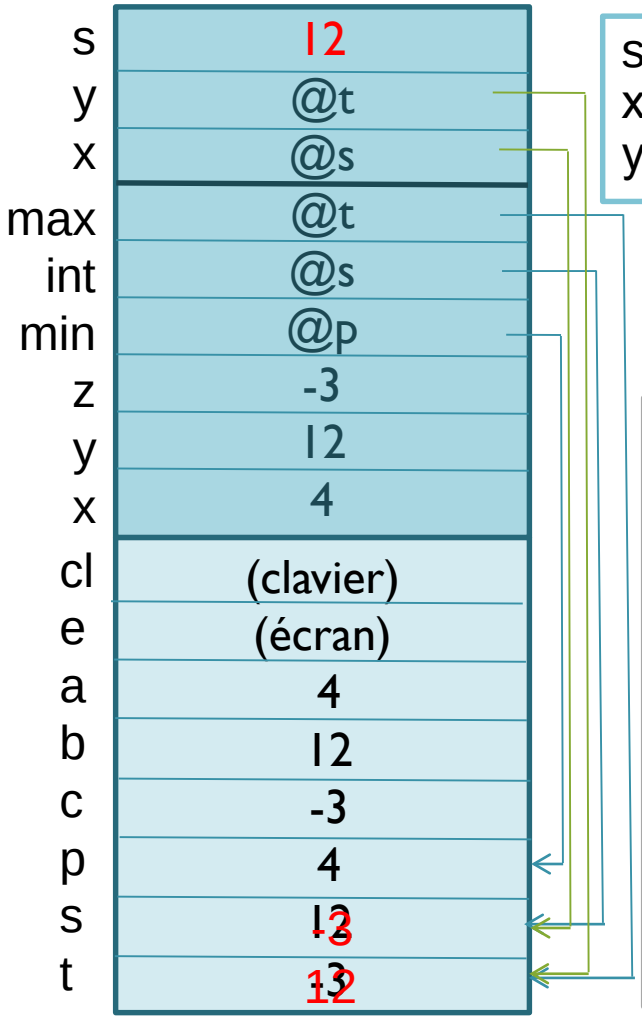
Appel de l'action classer3V

Instructions exécutées :

Création du contexte de échanger



Appel de l'action échanger



Instructions exécutées :

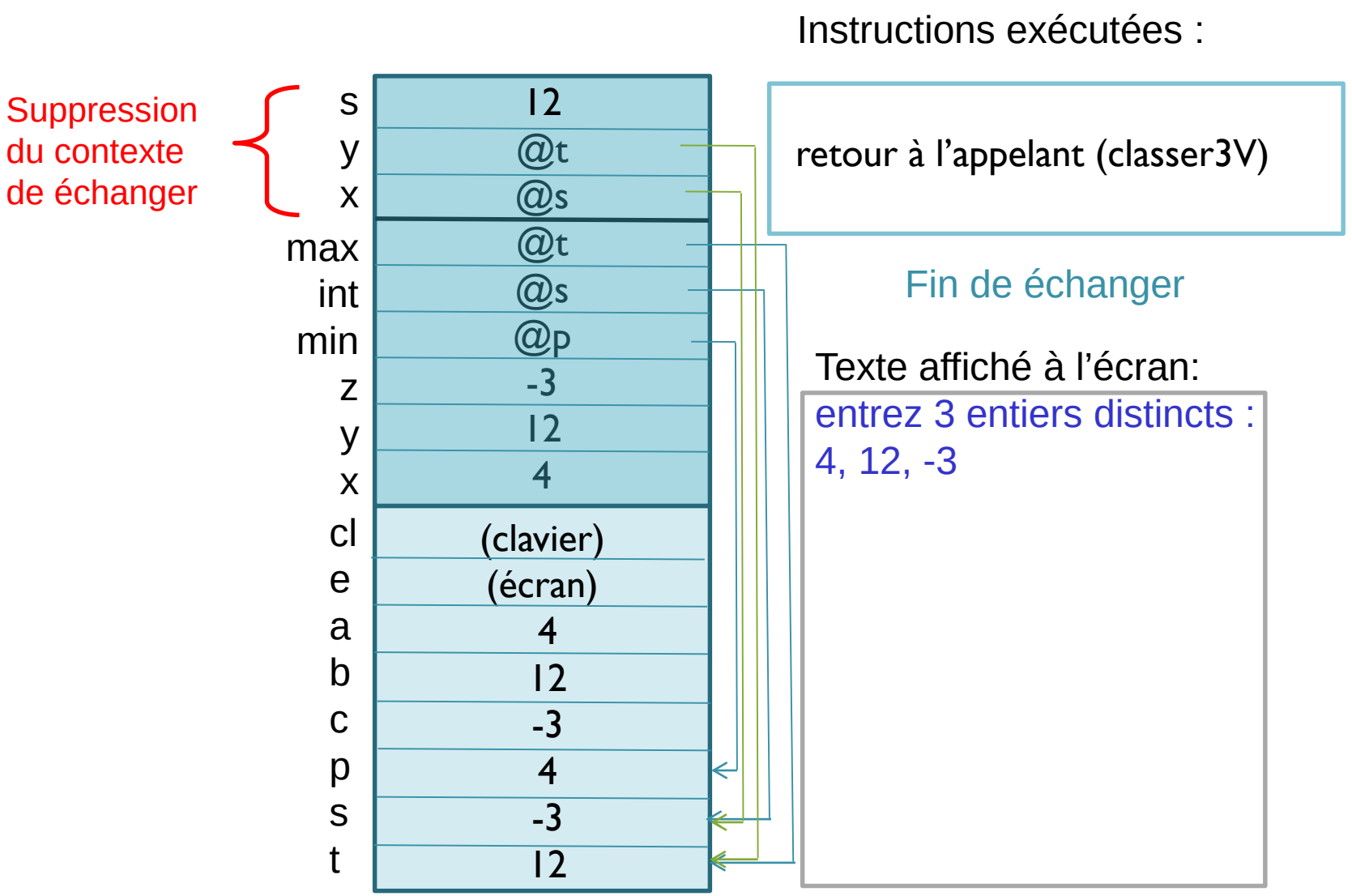
s ← x
x ← y
y ← s

Exécution de échanger

Texte affiché à l'écran:

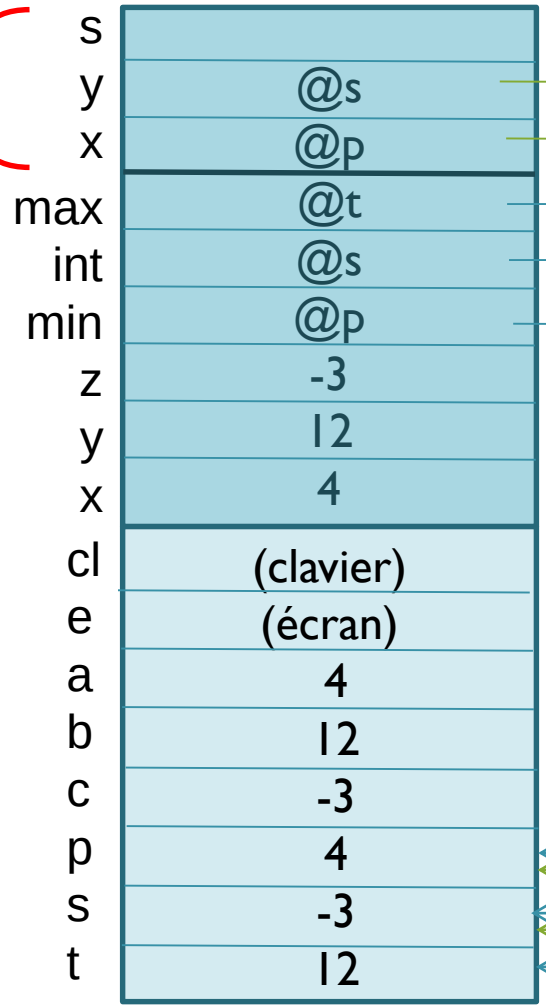
entrez 3 entiers distincts :
4, 12, -3

Appel de l'action échanger



Appel de l'action classer3V

Création du contexte de échanger



Instructions exécutées :

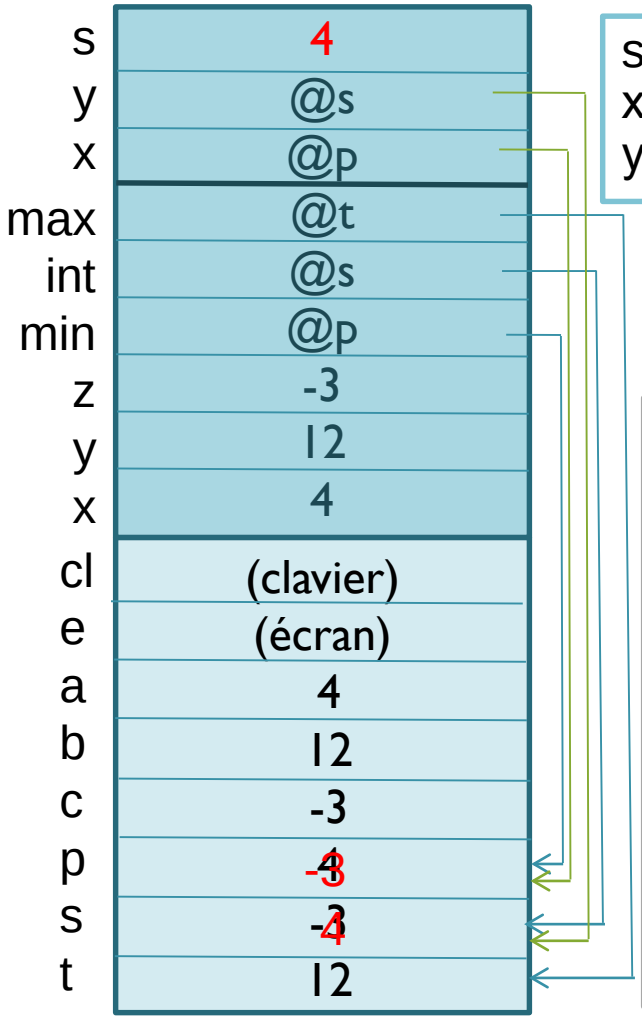
si min > int
alors échanger(min,int)
fsi

Appel de échanger

Texte affiché à l'écran:

entrez 3 entiers distincts :
4, 12, -3

Appel de l'action échanger



Instructions exécutées :

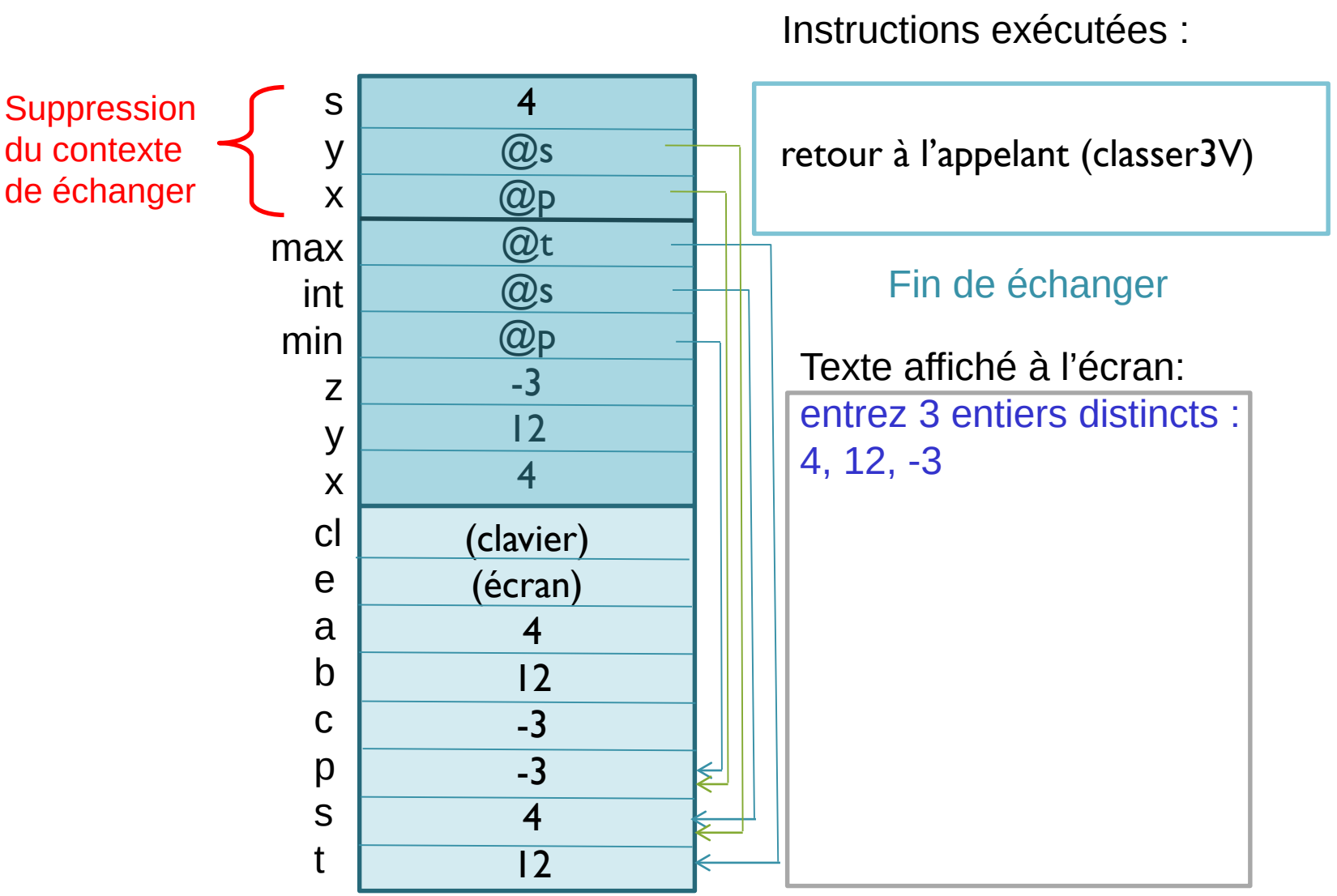
```
s ← x
x ← y
y ← s
```

Exécution de échanger

Texte affiché à l'écran:

entrez 3 entiers distincts :
4, 12, -3

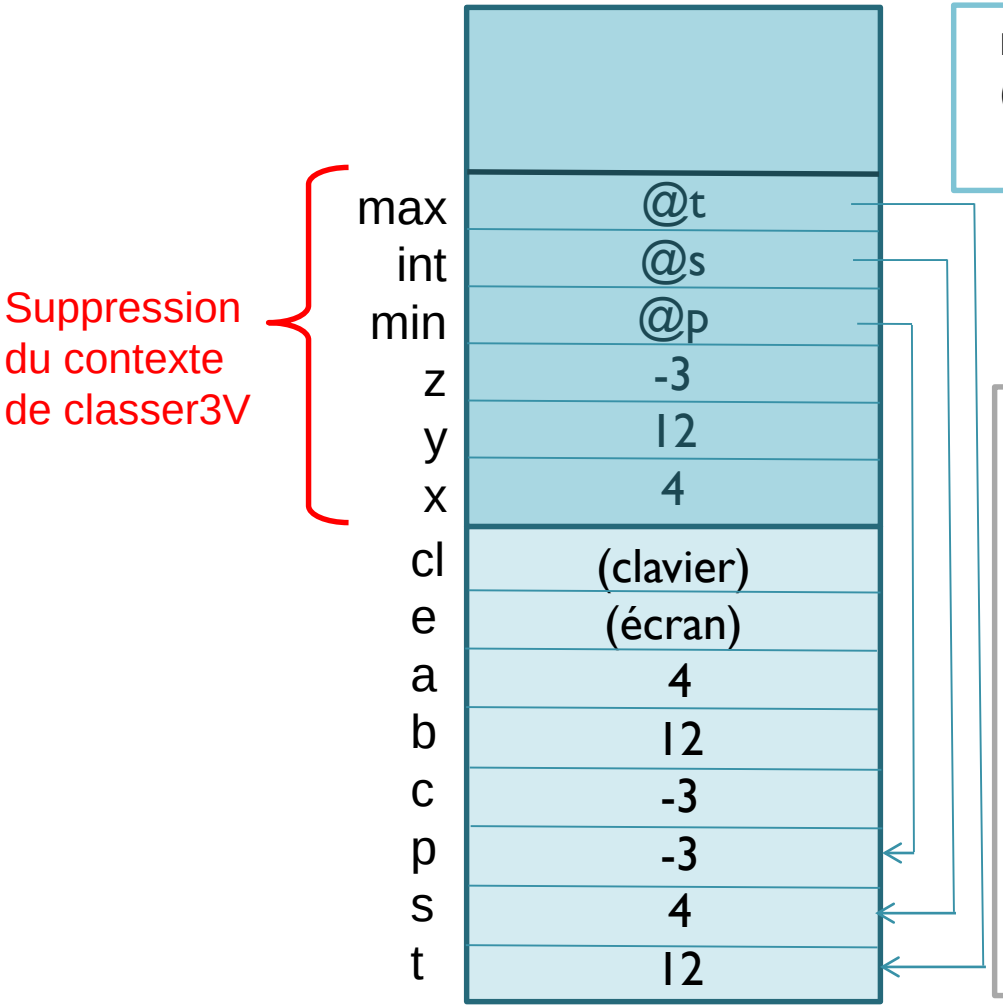
Appel de l'action échanger



Fin de l'action classer3V

Instructions exécutées :

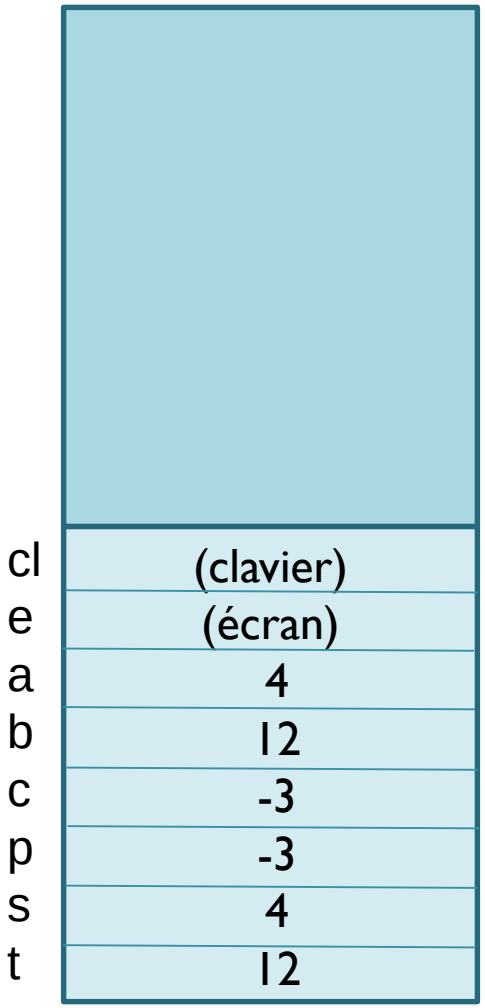
retour à l'appelant
(algorithme principal)



Texte affiché à l'écran:

entrez 3 entiers distincts :
4, 12, -3

Algorithme principal



Instructions exécutées :

```
e.afficher(p, '  ', s, '  ', t)
```

Fin de l'algorithme principal

Texte affiché à l'écran:

```
entrez 3 entiers distincts :
4, 12, -3
-3 4 12
```

Etude d'un objet fournissant un répertoire d'actions de tracés

- La machine-tracés
- Origine : la tortue de Seymour Pappert
- Wikipedia :

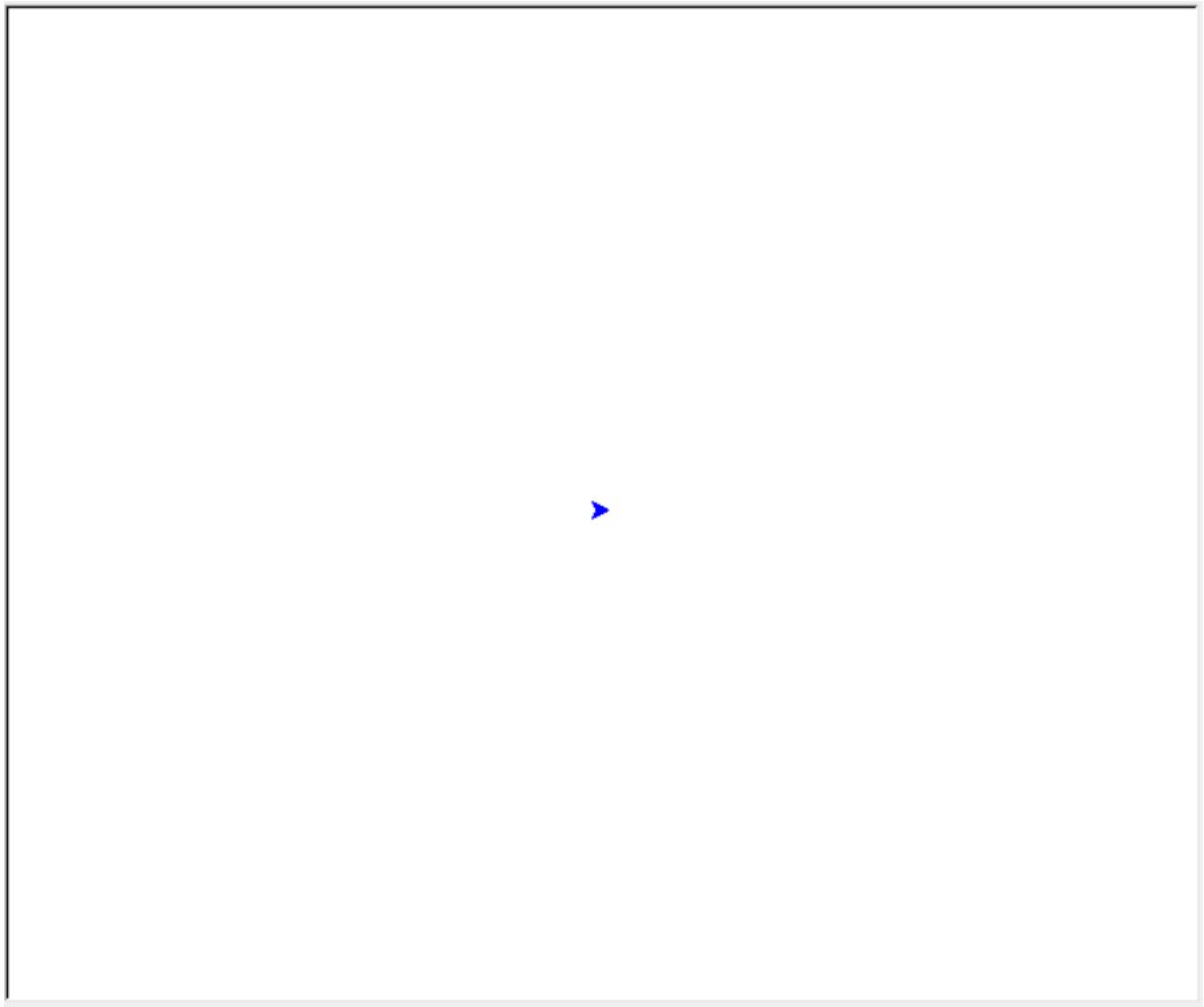
"*Seymour Papert, né le 29 février 1928 à Pretoria en Afrique du Sud et mort le 31 juillet 2016 à Blue Hill aux États-Unis, a été mathématicien, informaticien et éducateur au Massachusetts Institute of Technology (MIT). Il a été l'un des pionniers de l'intelligence artificielle, ainsi que l'un des créateurs du langage de programmation Logo.*"

Spécifications de la machine-tracés

La machine-tracés est un objet informatique caractérisé par :

- un écran formé de points pouvant être allumés ou éteints,
- une « plume » ou « tortue » permettant de réaliser des tracés sur l'écran à l'aide d'un répertoire d'actions.
- La machine-tracés fournit des actions prédéfinies permettant d'agir sur l'état de l'écran et de la plume

La machine-tracés



Etat de la machine-tracés

- écran : description de ce qui est affiché à l'écran (tracé)
- plume :
  - **cap** = angle en degrés, convention de la trigonométrie
  - position d'écriture (**pe**) : [haute, basse],
  - position dans le plan (**pp**) : un point

Actions de modification globale de l'écran et de la plume

action **vider**

// Effet : vide l'écran et positionne la plume au centre, en
// position haute, cap 0
// E.I. : écran et plume indifférents
// E.F. : écran vide, cap = 0, pe = haute, pp = (0,0)

action **effacer**

// Effet : vide l'écran sans modifier l'état de la plume
// E.I. : écran indifférent, cap = a_0, pe = p_0, pp = s_0
// E.F. : écran vide, cap = a_0, pe = p_0, pp = s_0

Actions de modification de la position d'écriture de la plume

action **lever**

// Effet : lève la plume

// E.I. : écran indifférent, cap = a_0, pe = p_0, pp = s_0

// E.F. : écran inchangé, cap = a_0, pe = haute, pp = s_0

action **baisser**

// Effet : baisse la plume

// E.I. : écran indifférent, cap = a_0, pe = p_0, pp = s_0

// E.F. : écran inchangé, cap = a_0, pe = basse, pp = s_0

Actions de modification de la plume dans le plan

action **avancer** (consulté x : réel) (ou **av**)

// Effet : avance la plume dans la direction du cap sur une longueur x
// E.I. : écran indifférent, cap = a_0, pe = p_0, pp = s_0 , x = x_0
// E.F. : si p_0 = haute, écran inchangé ; si p_0= basse, un segment de longueur x_0 est tracé depuis s_0 dans la direction a_0 ; soit s_1 l'extrémité de ce segment ; si x < 0, le segment est tracé dans la direction opposée soit 180+a_0 cap = a_0 , pe = p_0 , pp = s_1 , x = x_0

action **reculer** (consulté x : réel) (ou **re**)

// Effet : recule la plume dans la direction opposée au cap, sur une longueur x
// E.I. : écran indifférent, cap = a_0, pe = p_0, pp = s_0 , x = x_0
// E.F. : si p_0 = haute, écran inchangé ; si p_0= basse, un segment de longueur x est tracé depuis s_0 dans la direction a_0+180; soit s_1 l'extrémité de ce segment ; si x < 0, le segment est tracé dans la direction du cap a_0, pe = p_0, pp = s_1 , x = x_0

Actions de modification de la plume dans le plan

action **positionner** (consulté x : Point) (ou **pos**)

// Effet : positionne la plume au point x

// E.I. : écran indifférent, cap = a_0, pe = p_0, pp = s_0 , x = x_0

// E.F. : si p_0 = haute, écran inchangé ; si p_0= basse, un

// segment est tracé du point s_0 au point x_0

// cap = a_0 , pe = p_0 , pp = x_0 , x = x_0

Le type **Point** est défini comme suit :

type **Point** agrégat
x : réel
y : réel
fagrégat

Actions de modification du cap

action **gauche** (Consulté x : Angle) (ou **ga**)

// Effet : oriente la plume de x degrés à gauche par rapport au cap initial

// E.I. : écran indifférent, cap = a_0 , pe = p_0 , pp = s_0 , x = x_0

// E.F. : écran inchangé, cap = a_0 + x_0, pe = p_0, pp = s_0, x = x_0

action **droite** (Consulté x : Angle) (ou **dr**)

// Effet : oriente la plume de x degrés à droite par rapport au cap initial

// E.I. : écran indifférent, cap = a_0 , pe = p_0 , pp = s_0 , x = x_0

// E.F. : écran inchangé, cap = a_0 - x_0 , pe = p_0 , pp = s_0 , x = x_0

action **diriger** (Consulté x : Angle) (ou **dir**)

// Effet : oriente la plume dans la direction x

// E.I. : écran indifférent, cap = a_0 , pe = p_0 , pp = s_0 , x = x_0

// E.F. : écran inchangé, cap =x_0, pe = p_0 , pp = s_0 , x = x_0

Le type **Angle** est défini comme suit :

type Angle : réel entre -360 et 360

Utilisation de la machine-tracés

lexique principal
m : Machine-tracés

algorithme principal

```
m.vider
m.droite(120)
m.baisser
m.av(50)
m.re(50)
m.gauche(60)
m.av(50)
m.re(50)
m.lever
m.dir(0)
```

trouvez ce que fait cet algorithme en décrivant les états intermédiaires

Exemple d'action paramétrée de tracé

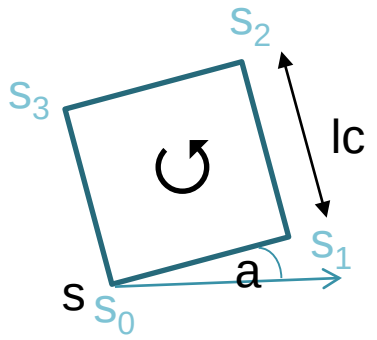
- Ecrire une action qui dessine un carré à l'aide d'une machine-tracés

action tracerCarré(consulté x : Carré, modifié m : Machine-tracés)
// Effet : trace le carré x à l'aide de la machine-tracés m
// E.I. : écran et plume indifférents
// E.F. : carré x tracé (ajouté à l'écran)

- Question : comment caractériser un carré ?

Caractérisation possible du carré

- Un point **s** correspondant à l'un des sommets
- La longueur **lc** des côtés du carré
- L'angle **a** d'orientation de tracé d'un côté à partir du sommet **s**
- Le sens du tracé ↻ ou ↺ (peut être implicite)



Tracé dans le sens trigonométrique

Lexique partagé

Carré : type agrégat
s : Point
lc : réel > 0
a : Angle
fagrégat

Réalisation de l'action de tracé

action tracerCarré(consulté x : Carré, modifié m : Machine-tracés)
// Effet : trace le carré x dans le sens trigonométrique, à l'aide de la machine-tracés m
// E.I. : écran et plume indifférents, x.s = s_0, x.lc=l_0, x.a=a_0
// E.F. : carré x tracé (ajouté à l'écran), pp = x.s= s_0, cap = x.a=a_0, pe = haute

Lexique de tracerCarré

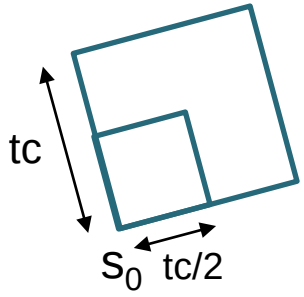
// x : Carré paramètre : carré à tracer
// m : Machine-tracés paramètre : machine-tracés utilisée

Algoritmme de tracerCarré

```
// écran et plume indifférents
m. lever ; m.positionner(x.s)  // pe = haute, pp = s0, cap = ?
m. diriger(x.a) ; m.baisser    // pp = s0, cap = a0, pe = basse
m.avancer(x.lc)                // 1er côté tracé, pp = s1, cap = a0, pe = basse
m.gauche(90)                   // 1er côté tracé, pp = s1, cap = a0+90, pe = basse
m.avancer(x.lc)                // 2ème côté tracé, pp = s2, cap = a0+90, pe = basse
m.gauche(90)                   // 2ème côté tracé, pp = s2, cap = a0+180, pe = basse
m.avancer(x.lc)                // 3ème côté tracé, pp = s3, cap = a0+180, pe = basse
m.gauche(90)                   // 3ème côté tracé, pp = s3, cap = a0+270, pe = basse
m.avancer(x.lc)                // 4ème côté tracé, pp = s0, cap = a0+270, pe = basse
m.gauche(90)                   // 4ème côté tracé, pp = s0, cap = a0+360=a0, pe = basse
m.lever                        // carré x tracé, pp = s0, cap = a0, pe = haute
```

Utilisation de l'action de tracerCarré

On veut tracer la figure suivante :



lexique principal
cl : Clavier // périphérique d'entrée
e : Ecran // périphérique de sortie
tc : réel > 0 // donnée : longueur côté carré extérieur
s : Point // donnée: sommet de départ
a : Angle // donnée : angle de tracé du 1er côté
ce : Carré // intermédiaire : carré extérieur
ci : Carré // intermédiaire : carré intérieur
m : Machine-tracés
action utilisée : tracerCarré

Algorithme principal
e.afficher("entrez les caractéristiques du carré extérieur:")
cl.saisir (s, tc, a) // s = s_0, tc = t_0, a = a_0
m.vider // écran vide, pp = <0,0>, cap = 0, pe = haute
ce.s ← s ; ce.lc ← tc ; ce.a ← a // ce = < s_0 , t_0, a_0 >
tracerCarré(ce,m) // carré extérieur tracé, pp= s_0, cap = a_0, pe = haute
ci.s ← s ; ci.lc ← tc/2 ; ci.a ← a; // ci = < s_0 , t_0/2 , a_0 >
tracerCarré(ci,m) // carré intérieur tracé, pp= s_0, cap = a_0, pe = haute

Simplification d'écriture : mise en facteur d'actions répétées

action tracerCarré(consulté x : Carré, modifié m : Machine-tracés)
// Effet : trace le carré x dans le sens trigonométrique, à l'aide de la machine-tracés m
// E.I. : écran et plume indifférents
// E.F. : carré x tracé (ajouté à l'écran), pp = x.s, cap = x.a, pe = haute

Lexique de tracerCarré
// x : Carré paramètre : carré à tracer
// m : Machine-tracés paramètre : machine-tracés utilisée

Algoritmme de tracerCarré

```
// écran et plume indifférents
m. Lever ; m.Positionner(x.s)  // pe = haute, pp = x.s, cap = ?
m. Diriger(x.a) ; m.Baisser    // pp = x.s, cap = x.a, pe = basse
m.avancer(x.lc)        // 1er côté tracé, pp = s1, cap = x.a, pe = basse
m.gauche(90)           // 1er côté tracé, pp = s1, cap = x.a+90, pe = basse
m.avancer(x.lc)        // 2ème côté tracé, pp = s2, cap = x.a+90, pe = basse
m.gauche(90)           // 2ème côté tracé, pp = s2, cap = x.a+180, pe = basse
m.avancer(x.lc)        // 3ème côté tracé, pp = s3, cap = x.a+180, pe = basse
m.gauche(90)           // 3ème côté tracé, pp = s3, cap = x.a+270, pe = basse
m.avancer(x.lc)        // 4ème côté tracé, pp = x.s, cap = x.a+270, pe = basse
m.gauche(90)           // 4ème côté tracé, pp = x.s, cap = x.a+360=x.a, pe = basse
m.lever                // carré x tracé, pp = x.s, cap = x.a, pe = haute
```

On répète 4 fois les Mêmes instructions

Simplification d'écriture : mise en facteur d'actions répétées

action tracerCarré(consulté x : Carré, modifié m : Machine-tracés)
// Effet : trace le carré x dans le sens trigonométrique, à l'aide de la machine-tracés m
// E.I. : écran et plume indifférents
// E.F. : carré x tracé (ajouté à l'écran), pp = x.s, cap = x.a, pe = haute
Lexique de tracerCarré
// x : Carré paramètre : carré à tracer
// m : Machine-tracés paramètre : machine-tracés utilisée
Algoritmme de tracerCarré
// écran et plume indifférents
m. Lever ; m.Positionner(x.s) // pe = haute, pp = x.s, cap = ?
m. Diriger(x.a) ; m.Baisser // pp = x.s, cap = x.a, pe = basse

On répète 4 fois les Mêmes instructions

répéter 4 fois
// à la $k^{ème}$ itétation, pp = s_{k-1}
m.avancer(x.lc) // $k^{ème}$ côté tracé, pp = $s_{k \bmod 4}$, cap = x.a+(k-1)*90, pe = basse
m.gauche(90) // $k^{ème}$ côté tracé, pp = $s_{k \bmod 4}$, cap = x.a+k*90, pe = basse
frépéter

m.lever // carré x tracé, pp = x.s, cap = x.a, pe = haute

Ecriture d'une action plus simple, mais dépendante du contexte

Idée : on trace le carré à partir de la position courante de la plume

- Le point de départ est implicite, l'ange aussi (cap courant)
- Le sens de tracé est aussi défini dans la spécification
- Le seul paramètre nécessaire est la longueur du côté du carré à tracer.

Réalisation de l'action simplifiée

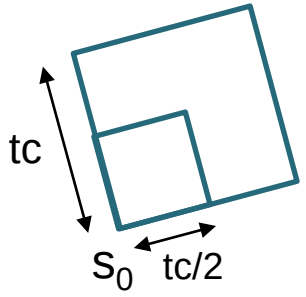
action tracerCarré2(consulté lc : réel > 0, modifié m : Machine-tracés)
// Effet : trace un carré de côté lc à partir de la position courante de la plume, à l'aide de la machine-tracés m
// E.I. : écran indifférent, pp = s_0, cap = a_0, pe = basse
// E.F. : carré de côté lc tracé, pp = s_0, cap = a_0, pe = basse

Lexique de tracerCarré2
// lc : réel > 0 paramètre: longueur du côté du carré à tracer
// m : Machine-tracés paramètre: machine-tracés utilisée

Algoritmme de tracerCarré2
// écran indifférent, pp = s_0, cap = a_0, pe = basse
répéter 4 fois
m.avancer(lc) // $k^{ème}$ côté tracé, pp = s_k, cap = a_0+(k-1)*90, pe = basse
m.gauche(90) // $k^{ème}$ côté tracé, pp = s_k, cap = a_0+k*90, pe = basse
frépéter
// carré de côté lc tracé, pp = s_0, cap = a_0, pe = basse

Utilisation de l'action simplifiée

On veut tracer la figure suivante :



lexique principal
  cl : Clavier // périphérique d'entrée
  e : Ecran // périphérique de sortie
  tc : réel > 0 // donnée : longueur côté carré extérieur
  s : Point // donnée: sommet de départ
  a : Angle // donnée : angle de tracé du 1er côté
  m : Machine-tracés
  action utilisée : tracerCarré2

Algorithme principal
  e.afficher("entrez les caractéristiques du carré extérieur:")
  cl.saisir (s, tc, a) // $s = s_0$, $tc = t_0$, $a = a_0$
  m.vider // écran vide, pp = <0,0>, cap = 0, pe = haute
  m.positionner(s) // écran vide, pp = s_0, cap = 0, pe = haute
  m.diriger(a) // écran vide, pp = s_0, cap = a_0, pe = haute
  m.baisser // écran vide, pp = s_0, cap = a_0, pe = basse
  tracerCarré2(tc, m) // carré extérieur tracé, pp=s_0, cap = a_0, pe = basse
  tracerCarré2(tc/2, m) // carré intérieur tracé, pp=s_0, cap = a_0, pe = basse

Lexique partagé (compléments)

- Dans certaines situations, le paramètre transmis à une action est toujours la même variable, dans l'exemple précédent, c'est le cas de m, la machine-tracés
- Il ne s'agit donc pas d'un véritable paramètre, puisque l'action ou la fonction **manipule toujours la même information**.
- Pour alléger l'écriture, on aimerait que ce paramètre puisse être implicite.

Lexique partagé (compléments)

- Une solution consiste à rendre cette information commune à l'algorithme principal et à l'action appelée en la définissant dans le **lexique partagé**.
- Lorsqu'une information est définie dans le lexique partagé, elle est visible et modifiable
  - de l'algorithme principal
  - des actions nommées
- On peut donc remplacer un tel paramètre par une variable partagée
- Attention ! Cette situation peut être source de nombreuses erreurs et nécessite une grande rigueur dans la spécification des actions qui partagent des informations.

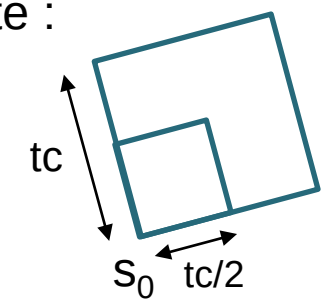
Exemple de variable partagée

Lexique partagé
 m : machine-tracés

action tracerCarré2(consulté lc : réel > 0)
// Effet : trace un carré de côté lc à partir de la position courante de la plume, à l'aide de la machine-tracés m (lexique partagé)
// E.I. : écran indifférent, pp = s_0, cap = a_0, pe = basse
// E.F. : carré de côté lc tracé, pp = s_0, cap = a_0, pe = basse
Lexique de tracerCarré2
 // lc : réel > 0 paramètre: longueur du côté du carré à tracer
Algoritmme de tracerCarré2
 // écran indifférent, pp = s_0, cap = a_0, pe = basse
 répéter 4 fois
  m.avancer(lc) // $k^{ème}$ côté tracé, pp = s_k, cap = a_0+(k-1)*90, pe = basse
  m.gauche(90) // $k^{ème}$ côté tracé, pp = s_k, cap = a_0+k*90, pe = basse
 frépéter
 // carré de côté lc tracé, pp = s_0, cap = a_0, pe = basse

Exemple de variable partagée

On veut tracer la figure suivante :



Lexique partagé
m : machine-tracés

lexique principal
cl : Clavier // périphérique d'entrée
e : Ecran // périphérique de sortie
tc : réel > 0 // donnée : longueur côté carré extérieur
s : Point // donnée: sommet de départ
a : Angle // donnée : angle de tracé du 1er côté

action utilisée : tracerCarré2

Algorithme principal
e.afficher("entrez les caractéristiques du carré extérieur:")
cl.saisir (s, tc, a) // s = s_0, tc = t_0, a = a_0
m.vider // écran vide, pp = <0,0>, cap = 0, pe = haute
m.positionner(s) // écran vide, pp = s_0, cap = 0, pe = haute
m.diriger(a) // écran vide, pp = s_0, cap = a_0, pe = haute
m.baisser // écran vide, pp = s_0, cap = a_0, pe = basse
tracerCarré2(tc) // carré extérieur tracé, pp=s_0, cap = a_0, pe = basse
tracerCarré2(tc/2) // carré intérieur tracé, pp=s_0, cap = a_0, pe = basse

Notion de durée de vie d'une variable

- Un variable définie dans le lexique partagé est créée au début de l'exécution du programme et détruite à la fin du programme ; sa durée de vie est donc égale à la durée de vie du programme
- Un variable définie dans un autre lexique est créée au début de l'exécution de l'algorithme qui est associé à ce lexique :
  - Lexique principal : durée de vie de l'algorithme principal
  - Lexique d'une fonction ou d'une action : durée de vie de l'appel de la fonction ou de l'action

Notion de durée de vie d'une variable

Lexique partagé
    a : entier // variable partagée appelée aussi « variable globale »

Créée au démarrage du programme

Lexique principal
    b, c : entiers // variables locales à l'algorithme principal
Algorithme principal
    cl.saisir(b)
    a ← f(b)
    e.afficher(a)

Créées au démarrage de l'algorithmes principal

Lexique de f
    x : entier // paramètre
    a, b : entier // variables locales à la fonction
Algorithme de f
    ….
    renvoyer(…)

Créées à l'appel de f

Détruites après le renvoi de la valeur calculée

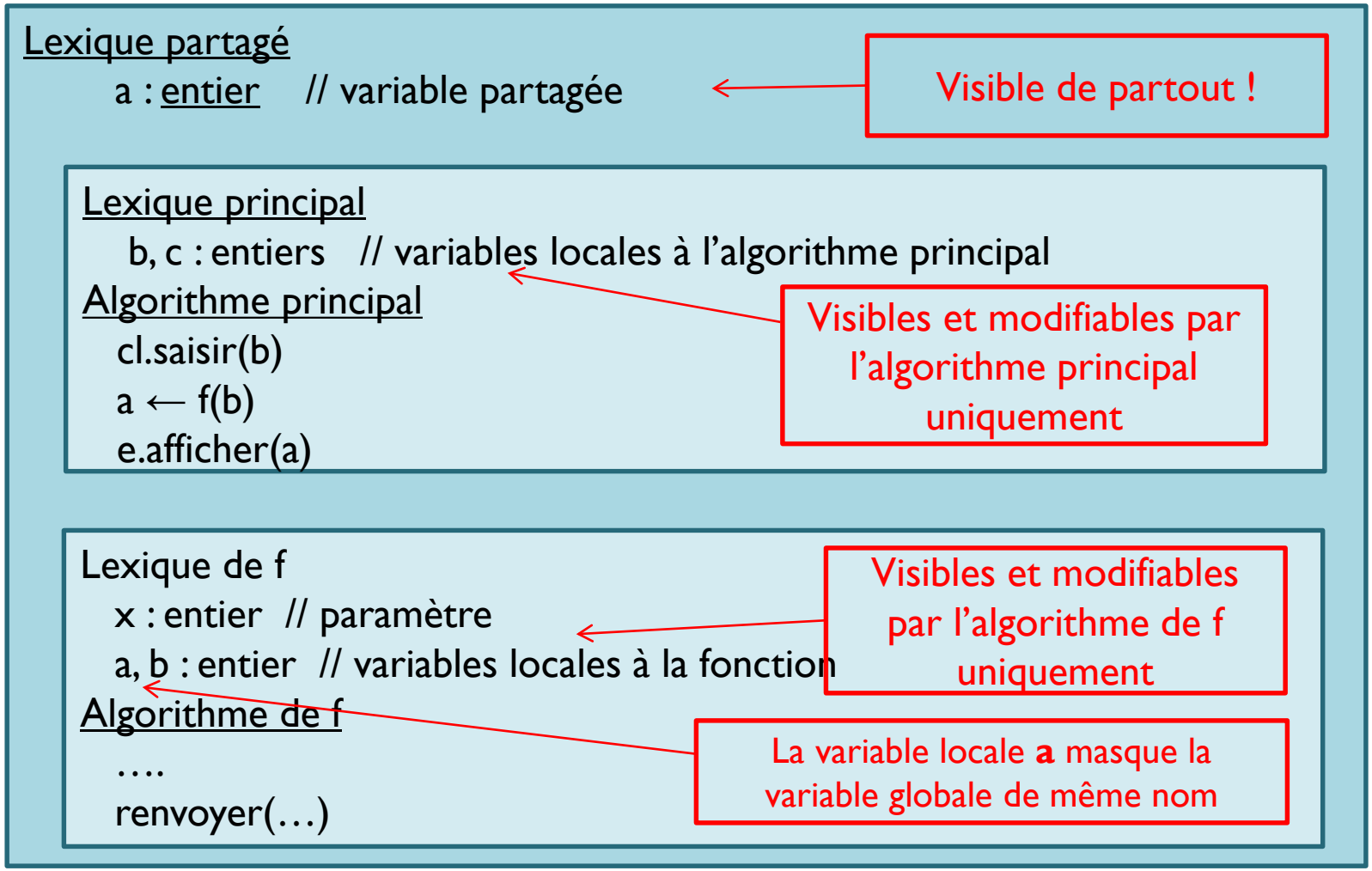
Notion de « portée » d'un variable

- Un variable définie dans le lexique partagé
  - est visible de l'algorithme principal et de toutes les actions et fonctions
  - est modifiable par l'algorithme principal et par les actions
  - Si une variable définie dans un lexique porte le même nom qu'une variable partagée, elle « masque » la variable partagée qui ne peut pas être atteinte.

Notion de « portée » d'un variable

- Une variable définie dans un autre lexique
  - est visible et modifiable uniquement par l'algorithme associé à ce lexique
  - les autres algorithmes ne peuvent pas la voir
- On peut donc définir des variables de même nom dans des lexiques différents, elles représenteront des informations différentes

Notion de « portée » d'un variable



Exercice 1

Dans l'algorithme ci-dessous, les assertions et les commentaires ainsi que les spécifications des actions manquent. Indiquez ce que fait l'algorithme et ce qu'il affiche à l'écran.

```
lexique partagé
  e : écran
```

```
lexique principal
  x : entier
  action utilisée : change1
algorithme principal
  e.afficher("début")
  x ← 0
  e.afficher("X = ", x)
  change1(x)
  e.afficher("X = ", x, " fin")
fin algorithme principal 1
```

```
action change1 (consulté x : entier)
lexique de change1
  // x : entier   paramètre
algorithme de change1
  e.afficher("début change1", "X= ", x)
  x ← x+1
  e.afficher("X = ", x, "fin change1")
fin algorithme de change1
```

Exercice 2

Dans l'algorithme ci-dessous, les assertions et les commentaires ainsi que les spécifications des actions manquent. Indiquez ce que fait l'algorithme et ce qu'il affiche à l'écran.

```
lexique partagé
   e : écran
   x : entier
```

```
lexique principal
   action utilisée : change2
algorithme principal
   e.afficher("début")
   x ← 0
   e.afficher("X = ", x)
   change2 ()
   e.afficher("X = ", x, " fin")
fin algorithme principal
```

```
action change2( )
lexique de change2
    x : entier
algorithme de change2
   e.afficher("début change2", "X= ", x)
   x ← 1
   e.afficher("X = ", x, "fin change2")
fin algorithme de change2
```

Exercice 3

Dans l'algorithme ci-dessous, les assertions et les commentaires ainsi que les spécifications des actions manquent. Indiquez ce que fait l'algorithme et ce qu'il affiche à l'écran.

```
lexique partagé
   e : écran
   x : entier
```

```
lexique principal
   y : entier
   action utilisée : change3
algorithme principal
   e.afficher("début")
   x ← 2 ; y ← 5
   e.afficher("x = ", x,  "y = ", y)
   change3(x)
   e.afficher("x = ", x, "y = ", y, "fin")
fin algorithme principal
```

```
action change3 (modifié y : entier)
lexique de change3
  // y : entier    : paramètre
algorithme de change3
     e.afficher("début change3","x= ",x, "y = ",y)
     x ← y +1
     y ← x+1
     e.afficher("X=", x, "Y=", y, "fin change3")
fin algorithme de change3
```

Exercice 4

Dans l'algorithme ci-dessous, les assertions et les commentaires ainsi que les spécifications des actions manquent. Indiquez ce que fait l'algorithme et ce qu'il affiche à l'écran.

```
lexique partagé
  e : écran
```

```
lexique principal
  a, b : entiers
  action utilisée : manipuler
algorithme principal
   e.afficher ("début")
   a ← 5
   e.afficher ("a = ", a)
   manipuler (a, b)
   e.afficher ("a = ", a, ", b = ", b)
   manipuler (b, a)
   e.afficher ("a = ", a, ", b = ", b)
   e.afficher ("fin")
fin algorithme principal
```

```
action manipuler( consulté x : entier,
                  élaboré y : entier)
lexique de manipuler
 // x : entier    : paramètre
 // y : entier    : paramètre
  z : entier
algorithme de manipuler
  e.afficher("début manipuler x= ", x, ", y = ", y,
                                    ", z = ", z)
  si x > 5
  alors  z ← 2
  sinon z ← - 3
  fsi
  y ← x - z
  e.afficher("fin manipuler x= ", x, ", y = ", y,
                                    ", z = ", z)
fin algorithme manipuler
```