



Université Paris XI
I.U.T. d'Orsay
Département Informatique
Année scolaire 2003-2004

Algorithmique : Volume 2

- Tableaux
- Sous-algorithmes
- Modularité

Cécile Balkanski, Nelly Bensimon, Gérard Ligozat

Tableaux



Ensemble de données du même type

Exemple de problème :

Saisir une suite de nombres, puis afficher cette suite après avoir divisé tous les nombres par la valeur maximale de la suite.

Nécessité de conserver les nombres en mémoire

variable contenant une valeur

val

132

variable contenant une collection de valeurs du même type

val

132 52 -57 -8902 -841 8100 -641

*Remarque : appeler cette variable **tabVal** plutôt que **val***

Les tableaux

Structure de données permettant d'effectuer un même traitement sur des données de même nature

tableau à **une**
dimension

--	--	--	--	--	--	--	--

tableau à **deux**
dimensions

Exemples d'applications

- Ensemble de valeurs entières, réelles, booléennes,....
- Ensemble de noms (type chaîne)
- Ensemble de caractères (type caractère)
- Ensemble d'adresses (type Adresse : nom, adresse, num téléphone)
- Ensemble d'ouvrages

Traitements opérant sur des tableaux

On veut pouvoir :

- **créer** des tableaux
- **ranger** des valeurs dans un tableau
- **recupérer, consulter** des valeurs rangées dans un tableau
- **rechercher** si une valeur est dans un tableau
- **mettre à jour** des valeurs dans un tableau
- **modifier** la façon dont les valeurs sont rangées dans un tableau (par exemple : les trier de différentes manières)
- effectuer des **opérations entre tableaux** : comparaison de tableaux, multiplication,...
- ...

Définition du type

nom du
tableau

tab

1 2 3 4 5 6

12	5	-78	2	-21	8
----	---	-----	---	-----	---

indice
du tableau

contenu
du tableau

unTab

1 2 3 4 5 6 7 8

p	i	s	c	i	n	e	s
---	---	---	---	---	---	---	---

Remarques :

- Indices : en général, démarrage à 1, **mais en C++, démarrage à 0**
- Nombre d'octets occupés : dépend du type des valeurs enregistrées

Déclaration d'un tableau

Exemple : déclaration d'un tableau pouvant contenir jusqu'à 35 entiers

variable *tab1* : tableau [1, 35] d'entiers

nom du tableau

mot clé

indices min et max
(taille)

type des éléments

Autre exemple : déclaration d'un tableau qui contiendra les fréquences des températures comprises entre -40°C et 50°C

variable *températures* : tableau [-40, 50] de réels

Les éléments
sont numérotés
de -40 à 50

Définition d'un type tableau

```
type <Nom> = <description>
```

Exemple : *déclaration d'un nouveau type Mot,
tableau de 10 caractères*

```
type Mot = tableau [1, 10 ] de caractères  
variables nom, verbe : Mot
```

Utilisation d'un tableau : par les indices

- Accès en lecture :
 - **afficher**(tabl[4]) *{le contenu du tableau à l'indice 4 est affiché à l'écran}*
- Accès en écriture :
 - tabl[3] ← 18 *{la valeur 18 est placée dans le tableau à l'indice 3}*
 - **saisir**(tabl[5]) *{la valeur entrée par l'utilisateur est enregistrée dans le tableau à l'indice 5}*
 - attention!

~~tabl ← 18~~

~~nom[2] ← 3~~

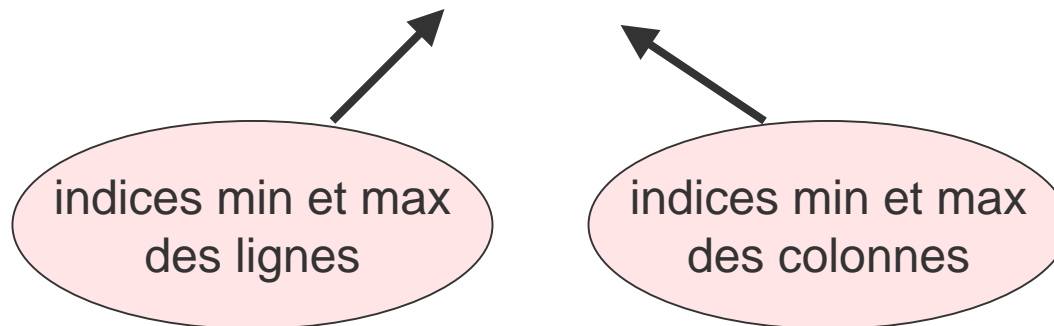
Tableaux à deux dimensions

	1	2	3	4	5	6	7
1	10	3	25	14	2	1	8
2	9	20	7	12	2	4	7

tableau à 2 lignes et 7 colonnes

- Déclaration

points : **tableau**[1,2 ; 1,7] d'entiers



Tableaux à deux dimensions (suite)

	1	2	3	4	5	6	7
1	10	3	25	14	2	1	8
2	9	20	7	12	2	4	7

- Accès en lecture :
 - **afficher**(points[1,7]) *{la valeur contenue en ligne 1 colonne 7 est affichée à l'écran}*
- Accès en écriture :
 - points[2,4] ← 36
 - **saisir**(points[2,4]) *{la valeur fournie est enregistrée en ligne 2, colonne 4}*

Saisir les valeurs d'un tableau 1D

Algorithme SaisieTableau

{remplit un tableau avec nbVal valeurs entières }

constantes (TailleMAX : entier) ← 100

variables nbVal, ind : **entier**

nombres : tableau [1, TailleMAX] d'entiers

début

afficher ("Combien de valeurs sont à saisir?")

saisir (nbVal)

si nbVal > TailleMAX

alors *{refuser la saisie : la capacité du tableau est dépassée}*

afficher ("trop de valeurs à saisir")

sinon pour ind ← 1 à nbVal **faire**

afficher ("Donner une valeur")

{valeur à ranger dans la ind^{ème} case du tableau}

saisir (**nombres[ind]**)

fpour

fsi

fin

Saisie avec « Drapeau »

Algorithme SaisieTableauAvecDrapeau

{remplit un tableau tant qu'il y a des caractères à ranger, dernier caractère : '\}'

constantes (TailleMAX : entier) \leftarrow 100

(DRAPEAU : caractère) \leftarrow '\'

variables nbLettres : **entier** *{nombre de caractères rangés dans lettres}*

lettres : **tableau** [1, TailleMAX] de **caractères**

unCar : **caractère**

début

nbLettres \leftarrow 0

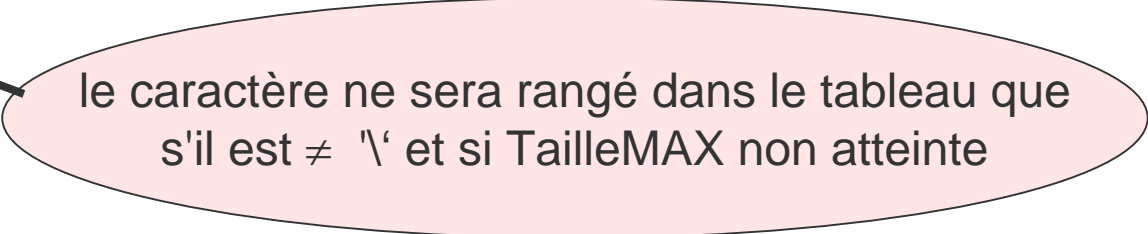
{saisie du premier caractère à ranger dans Lettres}

afficher (« Tapez un caractère, ou ", DRAPEAU, "pour arrêter la saisie. »)

saisir (unCar)

(Saisie avec Drapeau, suite)

```
{rangement du caractère saisi s'il est bon et saisie des caractères suivants}
tant que unCar ≠ DRAPEAU et nbLettres < TailleMAX faire
    nbLettres ← nbLettres + 1
    lettres[nbLettres ] ← unCar    {caractère rangé dans la nbLettresème
                                   case du tableau}
    afficher (" Tapez un autre caractère, ou ", DRAPEAU, "pour arrêter la saisie.")
    saisir (unCar )                {saisie du caractère suivant}
ftq
{test de sortie de boucle}
si unCar = DRAPEAU
    alors afficher ("Valeurs saisies intégralement.")
    sinon afficher ("Trop de caractères à saisir, plus de place ! ")
fsi
fin
```



le caractère ne sera rangé dans le tableau que s'il est ≠ \ ' et si TailleMAX non atteinte

Remarque : si unCar est différent de DRAPEAU, on est certainement sorti de la boucle parce que nbLettres est égal à TailleMAX.

Simulation de la saisie

Attention !

- Le drapeau **ne doit PAS** être rangé dans le tableau
- Le test de sortie ne peut pas être remplacé par

```
    si nbLettres = TailleMAX
        alors  afficher ("Trop de caractères à saisir,
                        plus de place ! ")
        sinon  afficher ("Valeurs saisies intégralement.")
    fsi
```
- Ne pas confondre
 - taille maximale : **TailleMAX** (une constante)
 - taille effective : **nbLettres** (une variable)

Affichage d'un tableau

Algorithme SaisitEtAffiche

{saisit et affiche un tableau de caractères}

constantes *{voir transparents précédents}*

variables *{voir transparents précédents}*

début

{saisie du tableau : voir transparents précédents}

{affichage}

afficher ("Voici les", nbLettres, "caractères saisis dans le tableau :")

pour cpt ← 1 à nbLettres **faire**

afficher (lettres[cpt])

fpour

fin

ATTENTION
exécuter la boucle
seulement nbLettres fois!

Saisir les valeurs d'un tableau 2D

Algorithme SaisieTableau2D

{remplit un tableau à 2 dimensions }

constantes (TailleMAX : entier) \leftarrow 100

variables nbLignes, nbColonnes, indL, indC : **entiers**

nombres : **tableau** [1, TailleMAX ; 1, TailleMAX] **d'entiers**

début

afficher ("Combien de lignes?") ; **saisir** (nbLignes)

afficher ("Combien de colonnes?") ; **saisir** (nbColonnes)

si nbLignes > TailleMAX **ou** nbColonnes > TailleMAX

alors **afficher** ("trop de valeurs à saisir")

sinon **pour** indL \leftarrow 1 à nbLignes **faire**

pour indC \leftarrow 1 à nbColonnes **faire**

afficher ("Ligne" , indL, "colonne", indC, " : ")

saisir (nombres[indL indC])

fpour

fpour

fsi

fin

Simulation

Sous-algorithmes

1. **Motivation, définitions, et simulation**
2. **Analyse à l'aide de sous-algorithmes**
3. **Retour aux tableaux: procédures et fonctions de manipulation de tableaux**

Identifier le rôle d'un bloc d'instructions

Algorithme Puissances

variables uneVal, puissance : **réels**
 cpt, nbPuissances : **entiers**

début

afficher ("Donnez un nombre réel quelconque : ")

saisir (uneVal)

afficher ("Combien de puissances successives souhaitez-vous ? ")

saisir (nbPuissances)

{calcul des nbPuissances puissances successives de uneVal }

puissance \leftarrow 1

pour cpt \leftarrow 1 à nbPuissances **faire**

 | puissance \leftarrow puissance \times uneVal

 | **afficher** ("La", cpt, "ième puissance de", uneVal, "est", puissance)

fpour

fin

En simplifiant l'écriture...

Algorithme Puissances

variables **uneVal** : **réel**
 nbPuissances : **entier**

début

afficher ("Donnez un nombre réel quelconque : ")

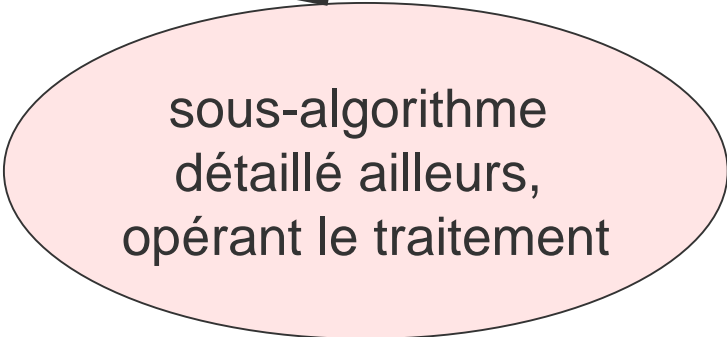
saisir (uneVal)

afficher ("Combien de puissances successives souhaitez-vous ?")

saisir (nbPuissances)

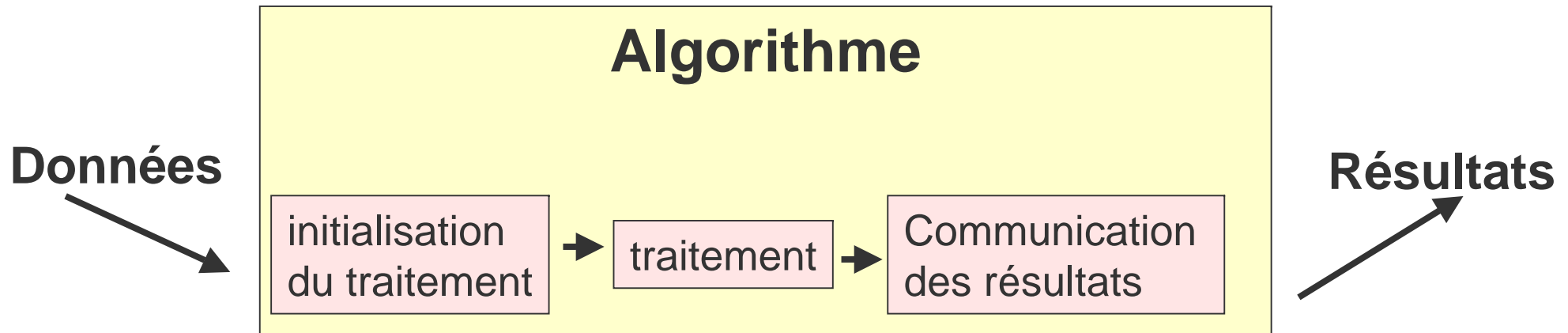
calculPuissances(uneVal, nbPuissances)

fin

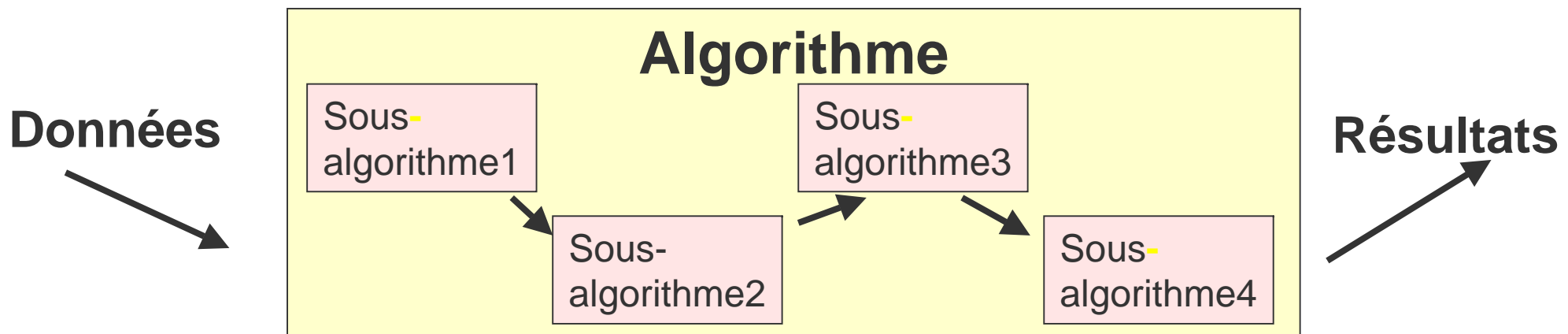


sous-algorithme
détaillé ailleurs,
opérant le traitement

Nouveau schéma d'un algorithme



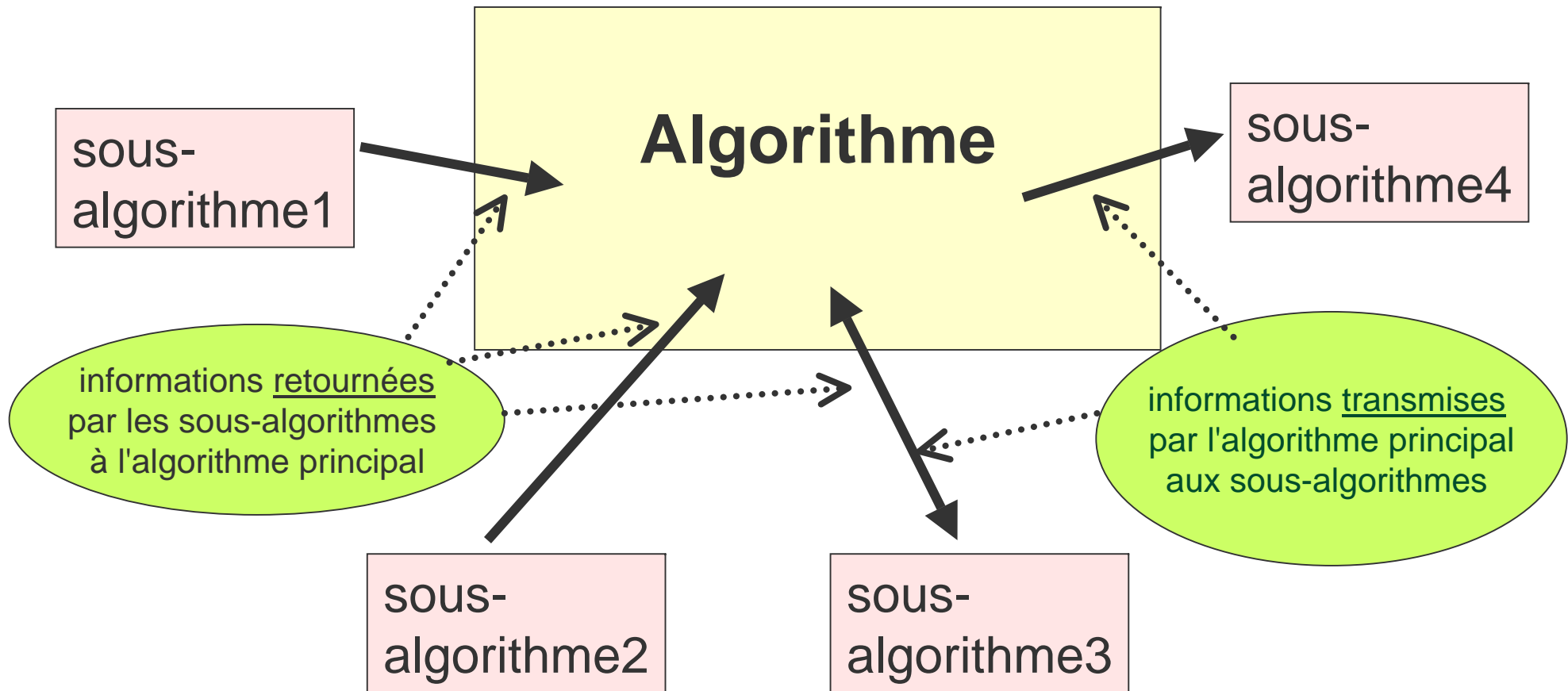
se ré-écrit en :



Sous-algorithmes

- Un algorithme **appelle** un sous-algorithme : cet algorithme *passe "momentanément" le contrôle* de l'exécution du traitement au sous-algorithme.
- Un sous-algorithme est conçu pour faire un traitement bien **défini**, bien **délimité**, si possible *indépendamment* du du contexte particulier de l'algorithme appelant.
- Remarque : un sous-algorithme peut en appeler un autre.

Comment les informations circulent...



Exemple

Algorithme Puissances

variables *uneVal* : réel
nbPuissances : entier

début

afficher (" ... ")

saisir (*uneVal*)

afficher (" ... ")

saisir (*nbPuissances*)

calculPuissances(*uneVal*, *nbPuissances*)

fin

Algorithme Puissances

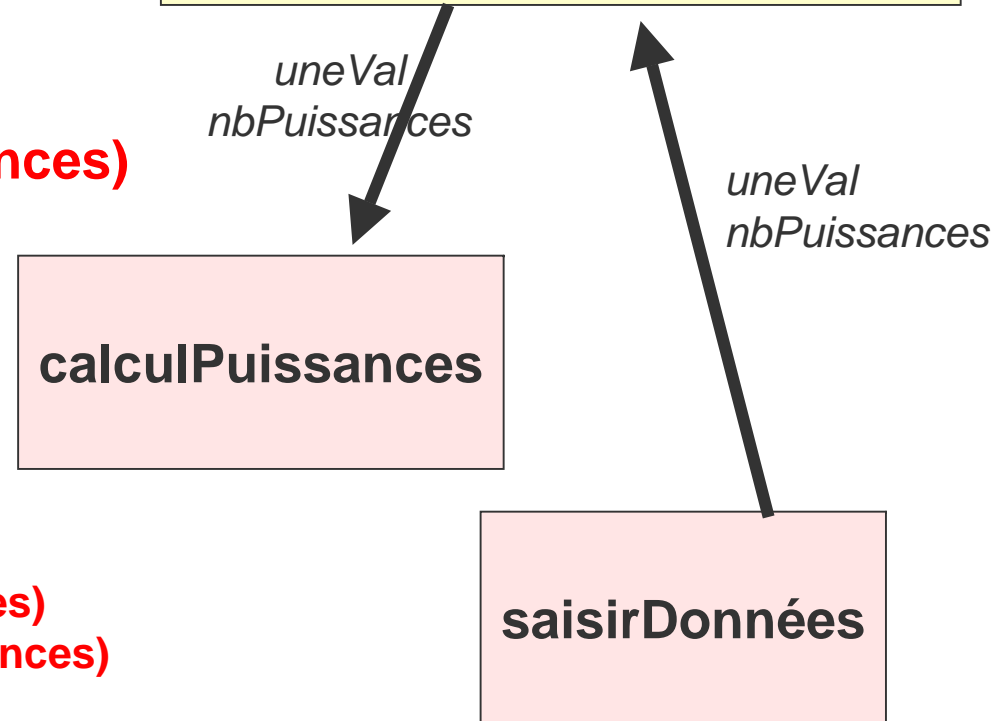
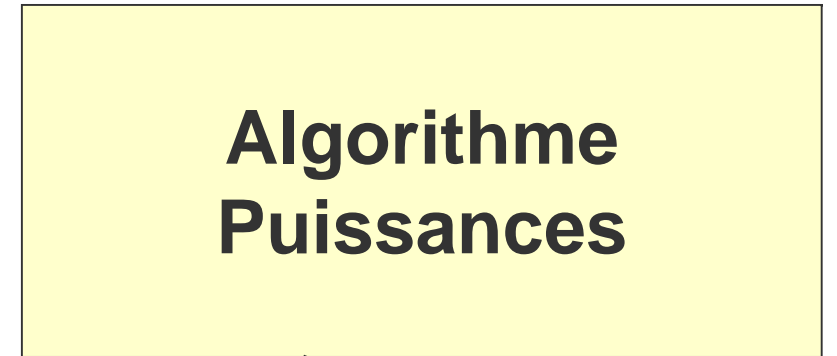
variables *uneVal* : réel
nbPuissances : entier

début

saisirDonnées(*uneVal*, *nbPuissances*)

calculPuissances(*uneVal*, *nbPuissances*)

fin



Communications d'informations

Algorithme

sous-algorithme

paramètre en « Donnée » noté (D)

paramètre en « Résultat » noté (R)

paramètre en « Donnée et Résultat » noté (D/R)

Paramètres et Arguments

Algorithme Puissances

variables uneVal : réels
nbPuissances : entier

début

...

calculPuissances(uneVal, nbPuissances)

...

fin

arguments de l'appel
de la procédure

Procédure calculPuissance(val, nb)

paramètre (D) val : réel
(D) nb : entier

début

....

fin

paramètres
de la procédure

Paramètres et Arguments



Au moment de l'appel du sous-algorithme, les valeurs des arguments sont affectés aux paramètres de statut (D) ou (D/R) :

- premier paramètre (si (D) ou (D/R)) \leftarrow premier argument
- deuxième paramètre (si (D) ou (D/R)) \leftarrow deuxième argument
- etc.

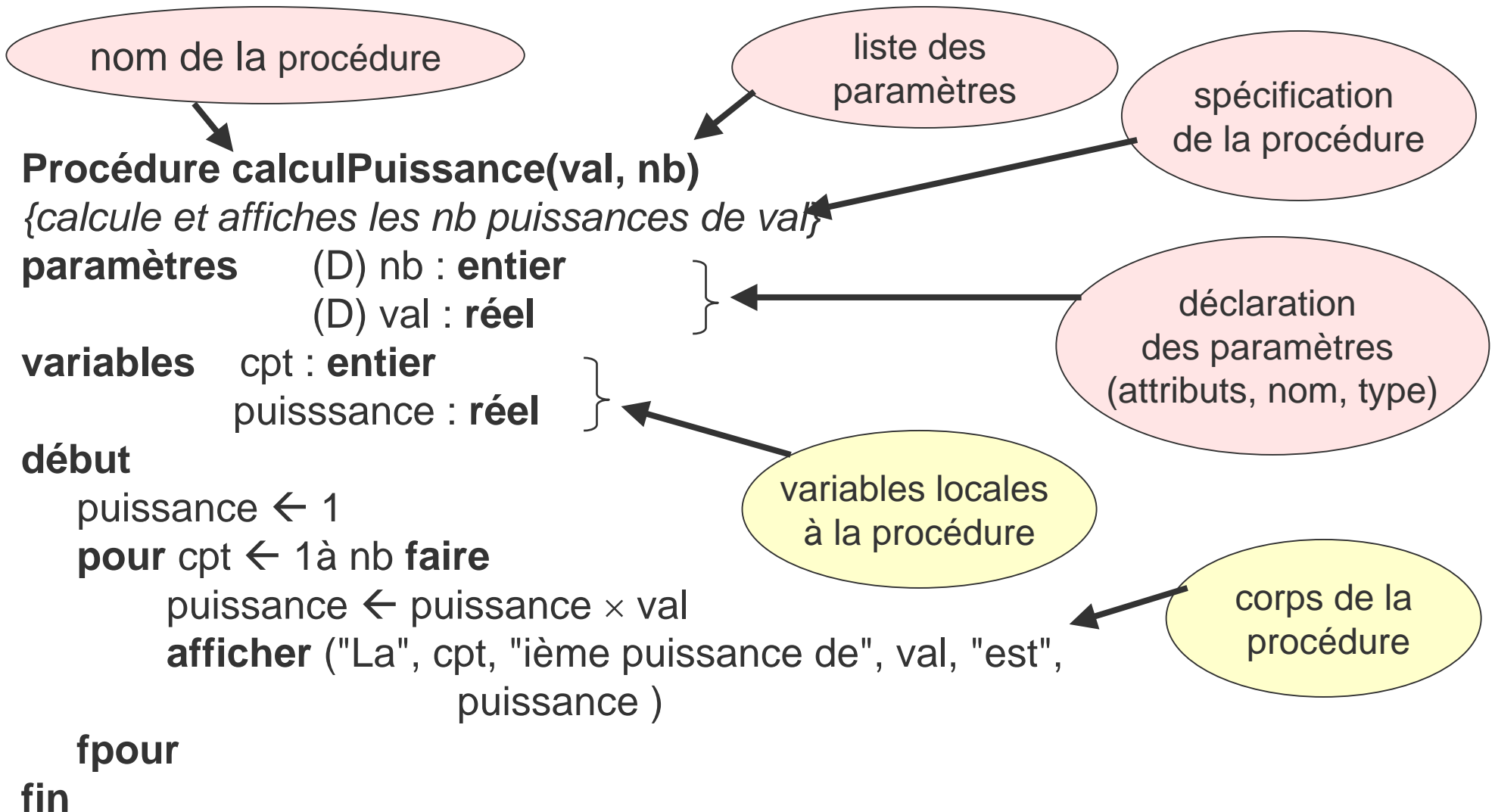
A la sortie du sous-algorithme, les valeurs des paramètres de statut (R) ou (D/R) sont affectés aux arguments correspondants :

- premier argument \leftarrow premier paramètre (si (R) ou (D/R))
- deuxième argument \leftarrow deuxième paramètre (si (R) ou (D/R))
- etc.

Paramètres et Arguments

- Le nombre de **paramètres** doit correspondre au nombre **d'arguments**.
- Le **type** du *kième* argument doit correspondre au type du *kième* paramètre.
- Le **nom** du *kième* argument *a*, de préférence, un nom différent de celui du *kième* paramètre.
- Un paramètre défini en "**Donnée**" doit correspondre à un argument qui possède une valeur dans l'algorithme appelant au moment de l'appel.
- Un paramètre défini en "**Résultat**" doit recevoir une valeur dans le sous-algorithme.

Procédure : formalisation syntaxique



Fonctions

Une fonction est un sous-algorithme qui **retourne** une valeur.

Algorithme surfaceRect

variables aire, long, larg : réels

début

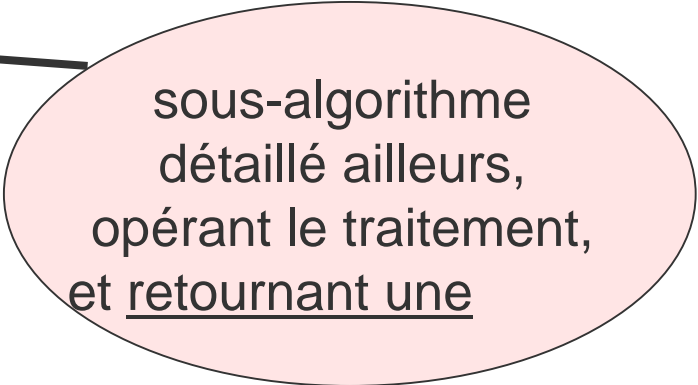
afficher ("Donnez la longueur et la largeur de votre rectangle : ")

saisir (long, larg)

aire ← surface(long, larg)

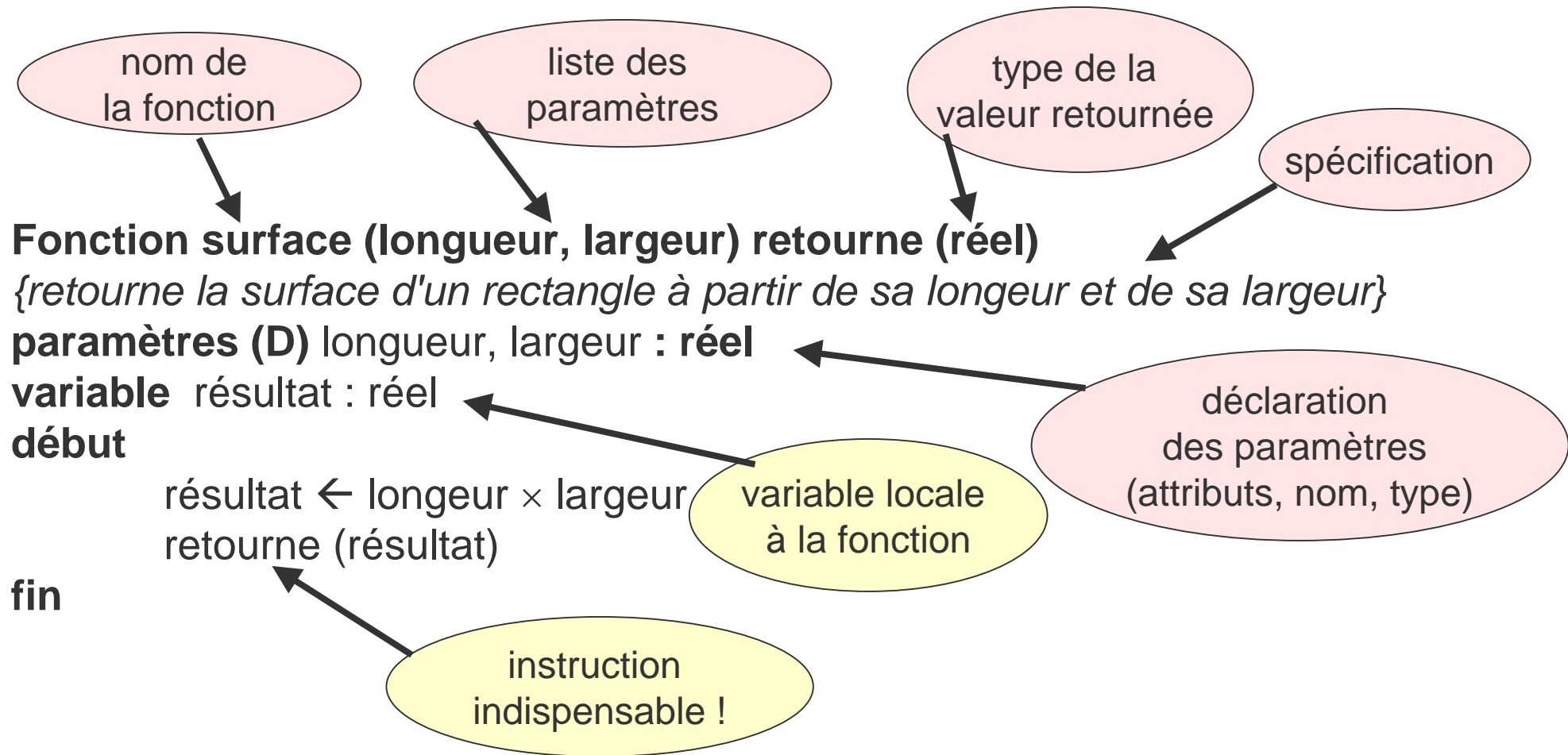
afficher ("Voici sa surface : ", aire)

fin



sous-algorithme
détaillé ailleurs,
opérant le traitement,
et retournant une

Fonction : formalisation syntaxique



Attention : pas de paramètre de statut (R) pour la valeur retournée

Comparaison : Fonctions et Procédures

- Définition :

Procédure surf1(long,larg, aire)

paramètres (D) long, larg : réel
(R) aire : réel

début

aire \leftarrow long \times larg

fin

Fonction surf2(long,larg) retourne réel

paramètres (D) long, larg : réel

variable aire : réel

début

aire \leftarrow long \times larg

retourne (aire)

fin

- Utilisation (appel):

Algorithme Exemple

variables surface, longueur, largeur : réel

début

longueur \leftarrow 10 ; largeur \leftarrow 25

surf1(longueur, largeur, surface)

surface \leftarrow surf2(longueur, largeur)

fin

*{utiliser surf1
ou surf2}*

Caractéristiques des deux classes de sous-algorithmes

- Une **procédure**, comme une **fonction**, est appelée dans un algorithme par son nom suivi de la liste des arguments correspondant à sa liste de paramètres.
- Lors de l'appel d'une **fonction**, la valeur retournée doit être exploitée dans une instruction (affectation, expression arithmétique ou logique, affichage, ...).
- Dans une **fonction**, la valeur retournée explicitement remplace le paramètre d'attribut résultat (R) auquel on veut donner une importance particulière.
- Dans une procédure comme dans une fonction, la liste des paramètres peut être **vide**.

Simulation de sous-algorithmes

Algorithme calculNotes

{calcule la moyenne des notes saisies }

variables note, somme, moyenne : **réel**
 nbnotes : **entier** ; ok : **booléen**

début

somme \leftarrow 0 ; nbnotes \leftarrow 0 *{initialisations}*

afficher ("Donnez la première note (note négative pour terminer)")

saisir(note)

tant que note \geq 0 **faire** *{traitement : saisies des notes et calcul du total}*

 | **compter**(note, somme, nbnotes)

 | **afficher** ("Donnez la note suivante (note négative pour terminer) ")

 | **saisir**(note)

ftq

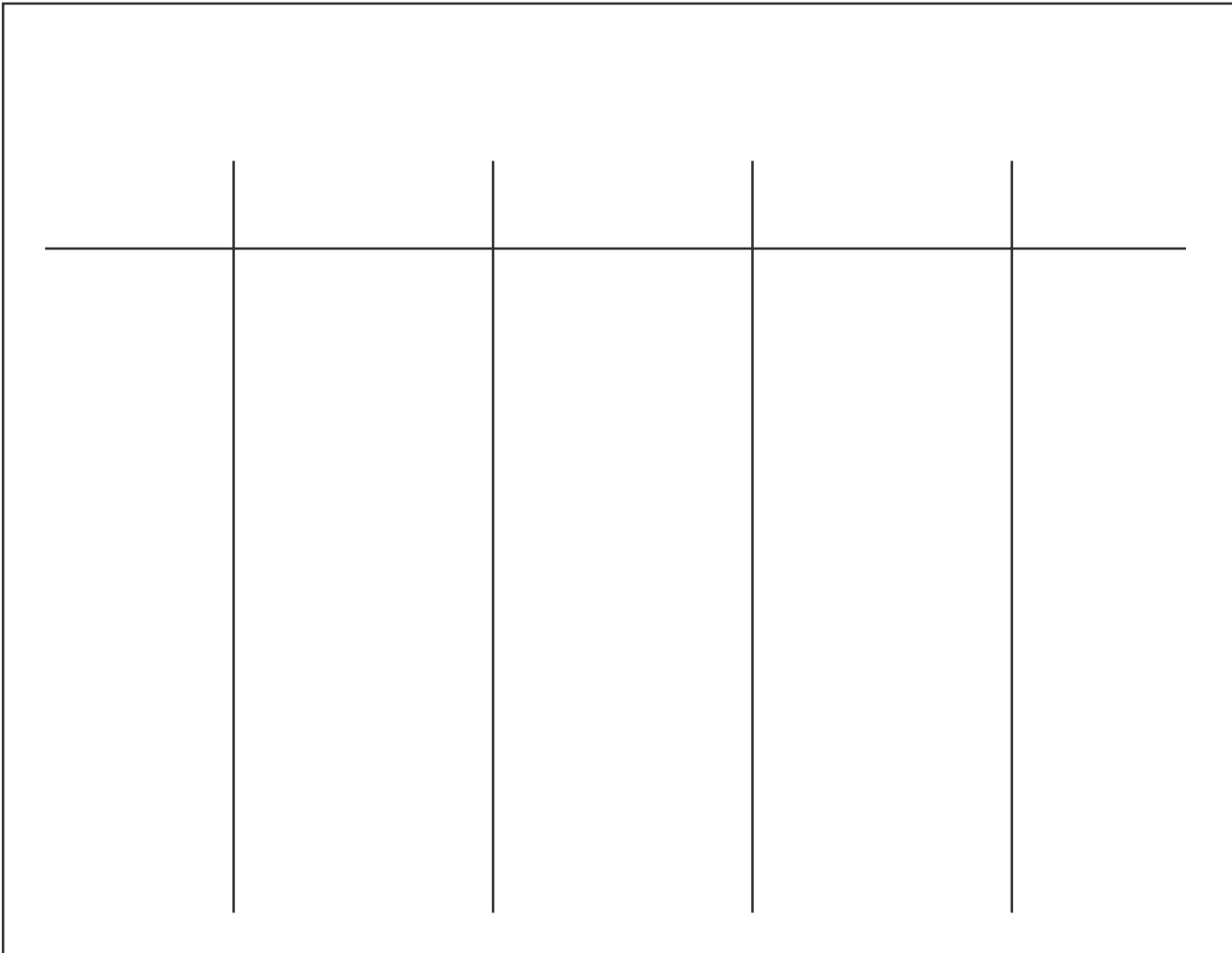
ok \leftarrow **calculMoyenne**(somme,nbnotes,moyenne) *{calcul moyenne}*

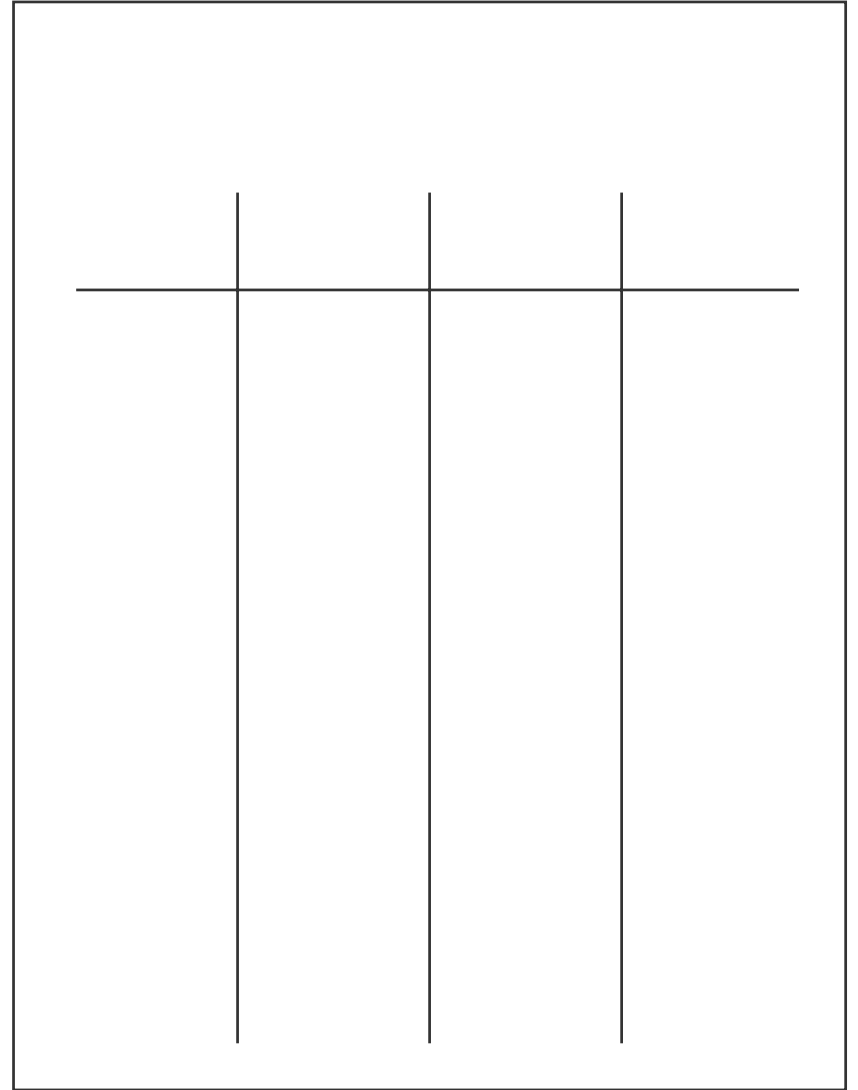
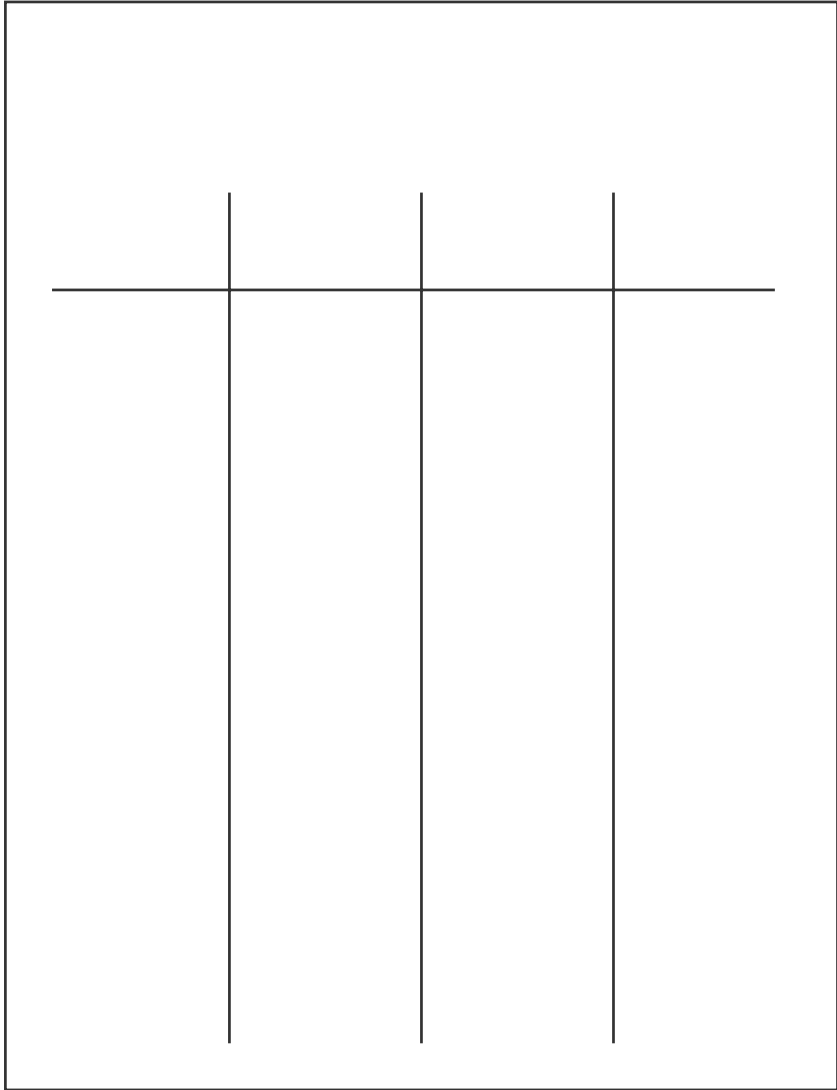
si ok *{affichage résultat}*

alors afficher("La moyenne des ", nbnotes, "notes est", moyenne)

sinon afficher("Pas de notes, pas de moyenne!")

fin





Procédure compter(uneNote, laSomme, nb)

{Ajoute uneNote à laSomme et incrémente nb}

paramètres

début

Fonction calculMoyenne(laSomme, nb, laMoyenne) retourne booléen

{Calcule la moyenne des nb notes dont le total est laSomme; retourne vrai si calcul possible, faux sinon}

paramètres

variable

début

Un autre exemple

Algorithme UnTest

{Cet algorithme mesure votre capacité à suivre des appels de sous-algorithmes}

variables a,b,c : **entiers**

début

a ← 1

b ← 2

afficher ("Entrée : (a, b) = (" , a , b , ")")

test1(a,b,c)

afficher ("Sortie : (a, b, c) = (" , a , b , c, ")")

fin

Procédure test1(x, y, z)

paramètres (D) x : entier
(D/R) y : entier
(R) z : entier

début

x ← x + 1

y ← y + 1

z ← test2(x, y)

afficher ("fin test1 : (x, y, z) = ("
x, y, z, ")")

fin

Fonction test2(v1, v2) retourne entier

paramètres (D) v1 : entier
(D/R) v2 : entier

variable rés : entier

début

afficher ("début test2 : (v1, v2) = ("
v1, v2, ")")

v1 ← v1 + 1

v2 ← v2 + 1

rés ← v1 + v2

retourner(rés)

fin

Simulation

Sous-algorithmes



2. Analyse à l'aide de sous-algorithmes

Problème :

A partir

- du jour de la semaine (chiffre entre 1(=lundi) et 7(=dimanche)),
- et de la date présentée sous le modèle AAAAMMJJ,
afficher la date du jour "en clair"

Exemple :

à partir des données : 2 20031014

affichage :

"Nous sommes le mardi 14 octobre 2003"

2 hypothèses de travail :

- l'utilisateur ne saisit que des chiffres
- l'utilisateur peut se tromper sur la cohérence des données

Algorithme principal

Algorithme AffichageDate

{faire un affichage lisible d'une date}

variables

début

{saisie des données}

saisieJourSemaine(

saisieDate(

{extraction des 4 chiffres de gauche dans la date}

calculAnnée(

*{extraction des 5ème et 6ème chiffres de la date, et
conversion dans le mois correspondant}*

calculMois(

{extraction des 2 chiffres de droite dans la date}

calculJour(

{présentation de la date}

afficherDate(

fin

Analyse de la circulation d'informations

Algorithme AffichageDate

saisieDate

saisieJourSemaine

afficherDate

calculAnnée

calculMois

calculJour

Détail des procédures (1)

Procédure saisieJourSemaine(nom)

{saisit le numéro du jour et retourne le nom du jour dans nom }

paramètre (R) nom : chaîne

variable numJ : entier

début

répéter

afficher ("Donnez le numéro du jour dans la semaine (nombre compris entre 1 et 7) :")

saisir (numJ)

tant que numJ < 1 ou numJ > 7

nom ← nomJourSemaine(numJ)

fin

variable locale
à la procédure

sous-algorithme qui
retourne le nom du (num_J)ème
jour de la semaine

Détail des procédures (2)

Procédure saisieDate(laDate)

{saisit la date sous forme d'une chaîne, en vérifie la cohérence, puis retourne dans laDate la chaîne convertie en entier}

paramètre (R) laDate : entier

variable unJour : chaîne

début

répéter

afficher ("Donnez la date (sous la forme AAAAMMJJ) :")

saisir (unJour)

tant que non dateValide(unJour)

 laDate ← convertir(unJour)

fin

sous-algorithme
qui vérifie la cohérence
de la date

sous-algorithme
qui transforme la chaîne
unJour en un entier

Détail des procédures (3)

Procédure calculAnnée(laDate, année)

{isole l'année année dans la date laDate (4 chiffres de gauche)}

paramètres (D) laDate, (R) année :entier

début

année \leftarrow laDate div 10 000

fin

Procédure calculMois(laDate, leMois)

{isole le mois leMois dans la date (chiffres des milliers et des centaines)}

paramètres (D) laDate : entier

(R) leMois : chaîne

variables unMois : entier

début

unMois \leftarrow (laDate mod 10 000) div 100

leMois \leftarrow nomMois(unMois) 

fin

sous-algorithme
qui retourne le nom du
(unMois)ème mois

Détail des procédures (4)

Procédure calculJour(laDate, leJour)

{isole le jour leJour dans la date laDate (2 chiffres de droite)}

paramètres (D) laDate, (R) leJour :entier

début

leJour ← laDate mod 100

fin

Procédure afficherDate(nomJ, jour, mois, année)

{affiche la date de façon bien lisible}

paramètres (D) nomJ, (D) nomM : chaînes
(D) jour, (D) année : entiers

début

afficher (" Nous sommes ", nomJ, " le ", jour, " ", nomM , " ", année)

fin

Détail des fonctions (1)

Fonction nomJourSemaine(numJ) retourne (chaîne)

{retourne le nom du (numJ) ème jour de la semaine}

paramètres (D) numJ : entier

variable leJour : chaîne

début

selon numJ :

1: leJour ← "Lundi"

2: leJour ← "Mardi"

3: leJour ← "Mercredi"

4: leJour ← "Jeudi"

5: leJour ← "Vendredi"

6: leJour ← "Samedi"

7: leJour ← "Dimanche"

retourne (leJour)

fin

Remarque : même genre de fonction pour nomMois(numM)

Hiérarchie des appels de sous-algorithmes

Algorithme AffichageDate

Remarque : procédure ou fonction?

Fonction dateValide (leJour) retourne (booléen)

{...}

paramètres (D) leJour : chaîne

variable bon : booléen

début

... bon ← ...

retourne (bon)

fin

Procédure dateValide (leJour, bon)

{...}

paramètres (D) leJour : chaîne

(R) bon : booléen

début

...

bon ← ...

fin

À l'appel (dans procédure saisieDate) :

✓ version fonction : tant que non **dateValide(unJour)** faire

✗ version procédure : tant que non **dateValide(unJour, bon)** faire

Remarque : procédure ou fonction? (suite)

Fonction convertir (leJour) retourne (entier)

{...}

paramètres (D) leJour : chaîne

variable uneDate : entier

début

... uneDate ← ...

retourne (uneDate)

fin

Procédure convertir (leJour, uneDate)

{...}

paramètres (D) leJour : chaîne

(R) uneDate : entier

début

...

uneDate ← ...

fin

À l'appel (dans procédure saisieDate) :

✓ version fonction : **laDate ← convertir(unJour)**

✓ version procédure : **convertir(unJour, laDate)**

Remarque : procédure ou fonction? (suite)

```
Fonction saisieDate() retourne (entier)
{...}
variable uneDate : entier
début
    ... uneDate ← ...
    retourne (uneDate)
fin
```

```
Procédure saisieDate(uneDate)
{...}
paramètres (R) uneDate : entier
début
    ...
    uneDate ← ...
fin
```

À l'appel (dans algorithme DateEnClair) :

- ✓ version fonction : **date** ← **saisieDate()**
- ✓ version procédure : **saisieDate(date)**

Bilan :

Intérêt de la programmation modulaire

- **Permettre une analyse descendante d'un problème :**
 - identifier les différents traitements contribuant au travail demandé
 - organiser l'enchaînement des étapes
- **Permettre une mise au point progressive, module par module**
- **Faciliter la maintenance des programmes**
 - modifier le traitement lui-même sans changer le rôle particulier d'un module
- **Enrichir le langage algorithmique en ajoutant de nouveaux "mots" du langage**
 - notion de "boite à outils", de bibliothèques de composants logiciels réutilisables

Sous-algorithmes



3. Retour aux tableaux :

Procédures et fonctions de manipulation de tableaux

Algorithme TraiterTableau

{saisit puis effectue des traitements sur une tableau d'entiers}

constantes (MAX : entier) \leftarrow 100
 (DRAPEAU : entier) \leftarrow 12345

variables nbVal, uneVal, ind : **entiers**
 valeurs : **tableau** [1, MAX] de **entiers**
 ok : **booléen**

début

{remplissage du tableau}

{vérification de la saisie}

{affichage du contenu du tableau}

(suite Algorithme TraiterTableau)

{somme des éléments du tableau}

{recherche d'un élément dans le tableau}

Procédure saisieTableau (tab, nbElt, correct)

{remplit le tableau tab tant qu'il y a des valeurs à ranger; retourne tab, sa taille effective nbElt, et correct à vrai si pas de pb}

paramètres (R) tab : **tableau** [1, MAX] de **entiers**

 (R) nbElt : **entier** ;

 (R) correct : **booléen**

variables uneVal : **entier**

début

afficher ("Donnez une valeur, ou bien", DRAPEAU, "pour arrêter la saisie.")

saisir (uneVal)

 nbElt \leftarrow 0

tant que uneVal \neq DRAPEAU **et** nbElt < MAX **faire**

 nbElt \leftarrow nbElt + 1

 tab[nbElt] \leftarrow uneVal

afficher ("Donnez une autre valeur, ", DRAPEAU, "pour arrêter.")

saisir (uneVal)

ftq

 correct \leftarrow (uneVal = DRAPEAU) *{vrai si la dernière valeur saisie était*

fin

le drapeau}

Procédure afficheTableau (tab, nbElt)

{affiche les nb valeurs du tableau tab}

paramètres tab : **tableau** [1, MAX] de **entiers**
 nbElt : **entier**

variables ind : **entier**

début

Fonction sommeTab (tab, nbElt) retourne entier

{retourne la somme des nb éléments du tableau tab}

paramètres tab : **tableau** [1, MAX] de **entiers**
 nbElt : **entier**

variables ind, total : **entiers**

début

Fonction recherche (tab, nbElt, uneVal, ind) retourne booléen

{Recherche le premier indice où se trouve la valeur uneVal parmi les nbElt données du tableau tab; retourne Vrai si trouvé, et faux sinon. }

Paramètres tab : **tableau** [1, MAX] de **entiers**
 nbElt, uneVal : **entiers**
 ind : **entier**

variables trouvé : **booléen**
début

Fonction ajout(tab, nbElt, uneVal) retourne booléen

{Ajoute la valeur uneVal au tableau tab de nb éléments. retourne vrai si Possible, sinon faux}

paramètres tab : **tableau** [1, MAX] de **entiers**

 nbElt : **entier**

 uneVal : **entier**

variable possible : **booléen**

début

Exemple de traitements avec plusieurs tableaux

Algorithme copieTableau

{Recopie un tableau dans un autre tableau en changeant le signe des valeurs }

constante (MAX : entier) \leftarrow 100

variables nbV, ind : **entier** ;

 tab1, tab2 : **tableau** [1, MAX] de **réels**

début

{remplissage du tableau tab1}

saisieTableau(tab1, nbV, ok)

{vérification de la saisie} ...

{remplissage du tableau tab2}

pour ind \leftarrow 1 à nbV **faire**

tab2 [ind] \leftarrow - tab1 [ind]

fpour

{suite du traitement}

 ...

fin

Comparaison de deux tableaux

Algorithme CompareTableaux

{Vérifie l'identité de deux tableaux}

constante **(MAX : entier) ← 100**

variables nbV1, nbV2, ind : **entiers**

 égaux, mêmeNbVal, ok : **booléens**

 tab1, tab2 : **tableau** [1, MAX] de **réels**

début

{remplissage du tableau tab1}

saisieTableau(tab1, nbV1, ok)

{vérification de la saisie} ...

{remplissage du tableau tab2}

saisieTableau(tab2, nbV2, ok)

{vérification de la saisie} ...

(Comparaison de deux tableaux, suite)

{Traitement : comparaison des deux tableaux}

mêmeNbVal \leftarrow nbV1 = nbV2

Si mêmeNbVal

alors *{parcours en parallèle tant qu'on ne trouve pas une différence}*

fsi

{Affichage résultats}

Retour au tableau à deux dimensions

10	3	25	14	2	1	8
9	20	7	12	2	4	7

constantes MAXLigne ..., MAXColonne ...

type T2D = **tableau**[1,MAXLigne ; 1,MAXColonne]
d'entiers

variable monTab : T2D

Quelques procédures associées au type T2D

Procédure **initTab2D(val, tab)**

{affecte la valeur val à tous les éléments du tableau 2 dim Tab}

paramètres (D) val : entier
 (R) tab : T2D

variables
début

Simulation :

Procédure saisirTab2DavecDim(tab, nbrL, nbrC)

{effectue la saisie d'un tableau 2 dim à nbrL lignes et nbrC colonnes}

paramètres (R) tab : T2D
 (D) nbrL, nbrC : entiers

variables

début

Simulation :

Procédure afficherTab2D(tab, nbrL, nbrC)

{affiche toutes les valeurs d'un tableau 2 dim à nbrL lignes et nbrC colonnes}

paramètres (D) tab : T2D
 (D) nbrL, nbrC : entiers

variables
début

Procédure copierTab2D(tabSource, tabDest, nbrL, nbrC)

{copie toutes les valeurs de tabSource dans tabDest; ces deux tableaux sont superposables}

paramètres (D) tabSource : T2D
 (R) tabDest : T2D
 (D) nbrL, nbrC : entiers

variables
début

Fonction rechercheVal(val, tab, nbrL, nbrC, numL, numC) retourne booléen

*{recherche les coordonnées numL et numC de la valeur val dans un tableau
2 dim; retourne VRAI si trouvé, FAUX sinon}*

paramètres (D) val, nbrL, nbrC : entiers
 (D) tab : T2D
 (R) numL, numC : entiers

Algorithme Exemple

{Traitements sur un tableau à deux dimensions.}

variables monTab : T2D
 nbrL, nbrC, uneVal, ind : entiers
 trouvé : booléen

début

afficher(«Entrez le nombre de lignes (≤ » , MAXLignes, «) »)

saisir(nbrL)

afficher(«Entrez le nombre de colonnes (≤ » , MAXColonnes, «) »)

saisir(nbrC)

{saisies supposées correctes}

saisirTab2DavecDim(monTab, nbrL, nbrC)

afficherTab2D(monTab, nbrL, nbrC)

afficher(«Quelle valeur voulez vous rechercher ?»)

saisir(uneVal)

trouvé ← **rechercheVal**(uneVal, monTab, nbrL, nbrC, numL, numC)

si trouvé

alors afficher(uneVal, « se trouve à la ligne », numL,
 « et à la colonne », numC)

sinon afficher(uneVal « ne se trouve pas dans le tableau. »)

fin

Modularité :



Méthodologie d'analyse d'un problème
en langage algorithmique

Problème : écrire l'algorithme du jeu de Saute Mouton

Sur une ligne de NB cases, on place, à la suite et en partant de la gauche, des pions noirs puis des pions rouges séparés par une case vide unique. On pose autant de pions noirs que de pions rouges. La configuration de pions n'occupe pas nécessairement toute la ligne.

But du jeu :

Déplacer tous les pions rouges vers la gauche (respectivement tous les pions noirs vers la droite), la case vide occupant à la fin du jeu la case du milieu de la configuration comme au départ.

Exemple :

configuration initiale :



1 2 3 4 5 6 7 8 9 10

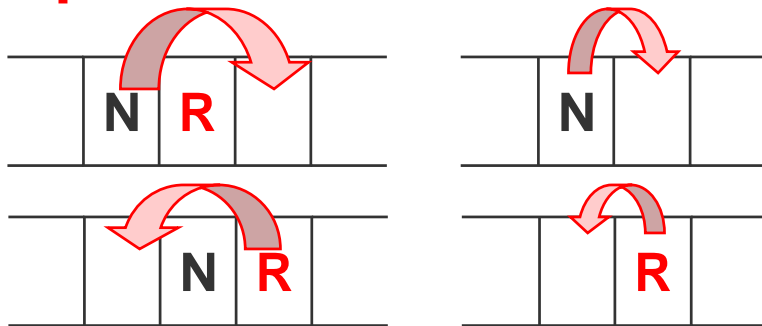
configuration finale gagnante :



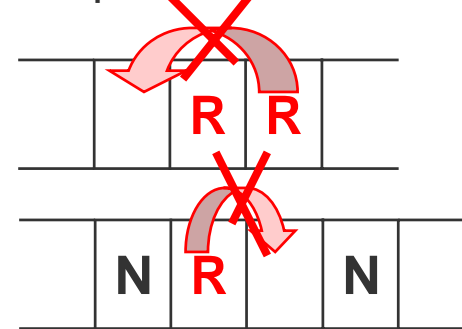
Règles du jeu:

- les pions **noirs** ne peuvent se déplacer que vers la droite
les pions **rouges** ne peuvent se déplacer que vers la gauche
- un pion **noir** peut être déplacé à droite dans la case vide :
 - si la case vide est juste à droite de ce pion
 - s'il lui suffit de sauter par dessus un seul pion rouge, c'est-à-dire si entre la case vide et lui il n'y a qu'un seul pion rouge.
- un pion **rouge** peut être déplacé à gauche dans la case vide :
 - si la case vide est juste à gauche de ce pion
 - s'il lui suffit de sauter par dessus un seul pion noir, c'est-à-dire si entre la case vide et lui il n'y a qu'un seul pion noir.

Exemple : coups possibles :



coups interdits :



Fonctionnement :

- A vous de jouer en donnant simplement la position du pion que vous jouez.
- La machine déplace le pion choisi si c'est possible.
- Le jeu s'arrête si vous avez gagné ou si vous avez atteint une situation de blocage. Dans ce cas vous avez perdu!

Comment lire un énoncé de problème dans le but d'écrire l'algorithme correspondant

- repérer les **données** proposées
- repérer les **résultats** attendus
- identifier les **contraintes** de la tâche
- définir les **traitements** à réaliser

• Structure de données :

Dans un premier temps, se demander quelles structure de données utiliser pour la représentation interne des données

- le plateau de jeu → **Plateau**: tableau de caractères

constante (NbMAX : entier) $\leftarrow 10$ *{nombre max de pions d'une couleur}*

type Plateau = tableau [1, 2 x NbMAX + 1] de caractères

variable jeu : Plateau

• Contraintes du jeu

Puis : bien comprendre les contraintes du jeu en explicitant les règles de façon plus formelle; inventorier les situations possibles.

si pion-rouge

alors si case-gauche est vide

alors déplacement-gauche

sinon si case-gauche-noire et case-suivante est vide

alors déplacement-saut-gauche

sinon si pion-noir

alors si case-à-droite est vide

alors déplacement-droite

sinon si case-droite-rouge et case-suivante est vide

alors déplacement-saut-droite

Déplacement \approx échange de valeurs entre la case vide et la case du pion à jouer.

• Hiérarchie des appels de sous-algorithmes

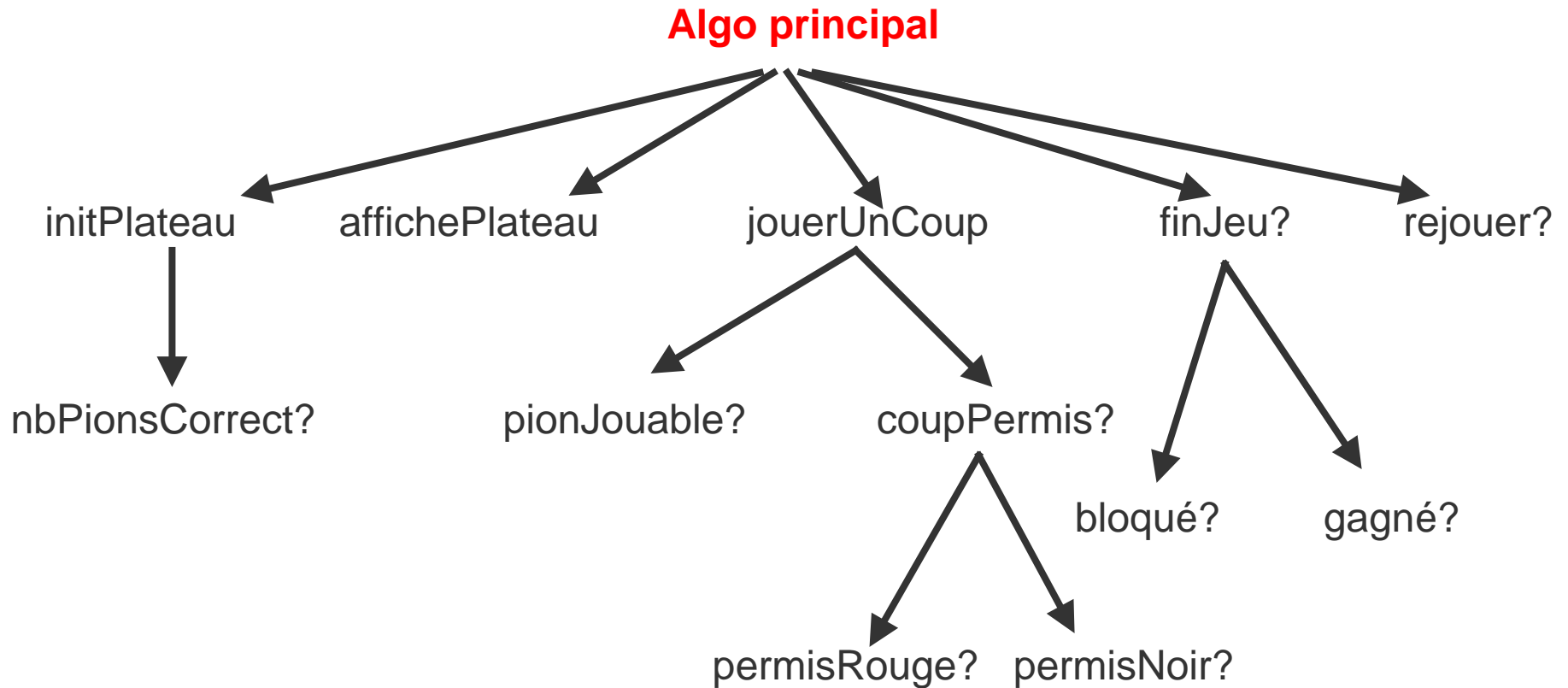
Puis : reformuler l'énoncé en tentant de planifier le travail, "décortiquer" les étapes successives du jeu.

→ décomposer la tâche en sous-tâches

Algorithme schématique

- initialiser le plateau de jeu
- afficher le plateau de jeu
- jouer un coup (si coup proposé est permis)
- rejouer si non fini
- si fini, proposer de refaire une partie

→ concevoir la hiérarchie des appels de sous-algorithmes



• Écriture de l'algorithme principal

(Une possibilité parmi d'autres)

Algorithme SauteMouton

{jeu de saute mouton}

constante (NbMAX : **entier**) \leftarrow 10 *{nombre maximum de pions d'une couleur}*

type Plateau = tableau [1, 2 x NbMAX + 1] de **caractères**

variables plateauJeu : **Plateau**

nbPions : **entier** *{nombre de pions d'une couleur}*

indVide : **entier** *{indice de la case vide}*

suite : **booléen** *{vrai si partie non terminée}*

début

répéter

initPlateau(plateauJeu,nbPions)

affichePlateau(plateauJeu,nbPions)

suite \leftarrow non finJeu?(plateauJeu,nbPions,indVide)

tant que suite faire

 jouerUnCoup(plateauJeu,nbPions,indVide)

 affichePlateau(plateauJeu,nbPions)

 suite \leftarrow non finJeu?(plateauJeu,nbPions,indVide)

ftq

tant que rejouer?

fin

• Spécification des sous-algorithmes

exemple:

Procédure initPlateau(tab,nb)

{demande le nombre nb de pions rouges à utiliser et vérifie que ce nombre est acceptable. Si oui, remplit le plateau tab de nb pions rouges et nb pions noirs, sinon demande un nouveau nombre.}

paramètres (R) tab : plateau

 (R) nb : entier

• Définition des sous-algorithmes

• Programmation

Attention !!!

Rien n'est figé, à tout moment on peut être amené à modifier des choix faits précédemment.

Comment faciliter la mise au point d'un programme

- donnez des **noms évocateurs** à vos variables, à vos constantes, à vos sous-algorithmes
- programmez **modulairement** : chaque bloc de l'algorithme remplit un sous-but élémentaire du problème général
- documentez votre travail : identifiez chaque bloc par un **commentaire explicatif**
- truffez l'algorithme d'affichages permettant de suivre à la **trace** l'exécution du programme correspondant
- mettez au point **module par module** votre programme, en construisant des **jeux d'essais** appropriés à chacun de ces modules
- prévoyez un **affichage de contrôle** des valeurs saisies aussitôt après la saisie

Des habitudes qu'il est vivement conseillé de prendre !

Construction d'un jeu d'essais pour la validation d'un algorithme

→ explorer tous les cheminements possibles à l'intérieur d'un algorithme

construction à partir de la modélisation informatique

→ inventorier tous les "cas de figure" des données soumises

prendre en compte :

- un cas général (ou plusieurs distingués)

- les cas extrêmes ("effets de bord")

construction à partir de la théorie

fin Volume 2