Informatique en CPGE
Lecture du programme

Présenté par D. ELGHANAMI
Pr. À l'EMI

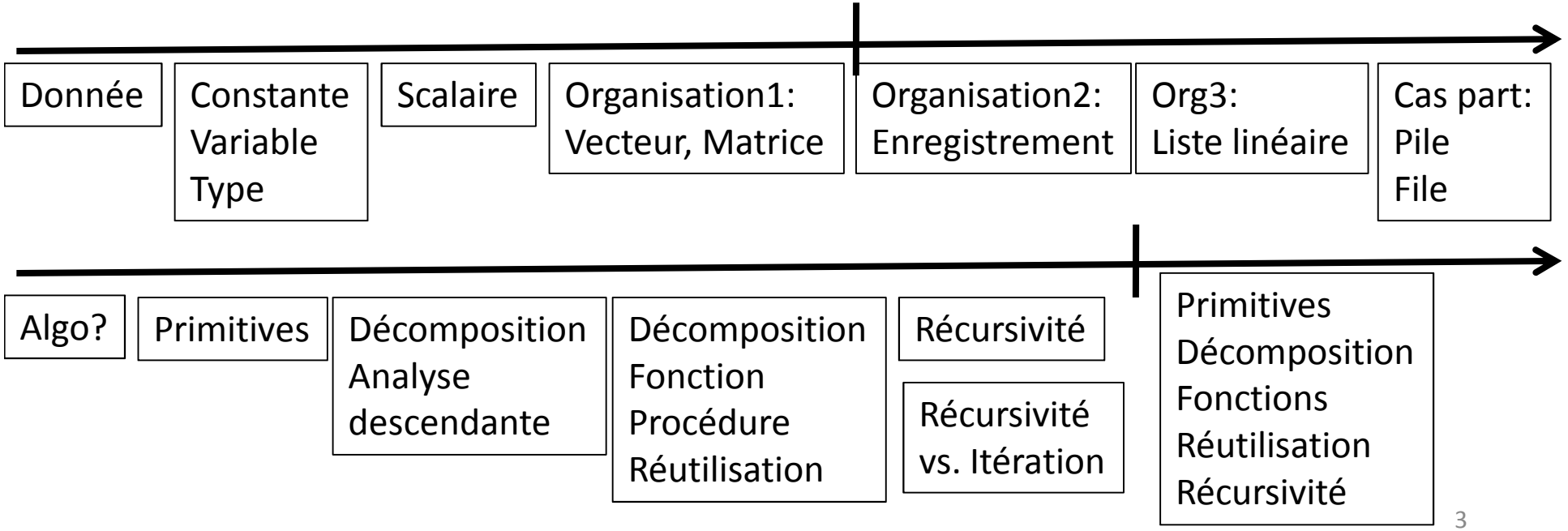
Marrakech le 16-18/12/2010

Les points abordés

1. Lecture du programme
2. Objectifs
3. Compétences
4. Situations
5. Limites du programme
6. Évaluations

Introduction

À travers une progression spiralée du programme, on montre comment la résolution de problèmes dans les différents chapitres sollicite et développe une démarche algorithmique de manière naturelle (logique, séquencer) aux différents moments du prog (à l'introduction d'une notion, en application d'une notion, en approfondissement d'une notion).



Objectif du programme

Apprendre à l'élève à résoudre des problèmes

→ Algorithmes

Définition : Suite précise de calculs nécessaires à la solution d'un problème dans une durée limitée.

1. Problème
2. Opérations de calcul
3. Durée limitée
4. Machine

La performance d'un algorithme

Représentation simplifiée de l'ordinateur

- Un PC contient une unité de calcul et des cases mémoires.
- Chaque case mémoire a une adresse (un entier).
- Chaque case mémoire contient une suite de 0, 1
- Le contenu d'une case mémoire peut être **interprété** comme :
    - nombre
    - caractère
    - boolean
    - instruction à effectuer

Problème $\xrightarrow{\text{Solution}}$ Algorithme = Données + Traitement

L'énoncé du problème spécifie la relation input / output souhaitée.

Données $\longrightarrow$ Instances des données $\longrightarrow$ Valeurs

<u>Questions :</u>

- La valeur est constante ou elle change
- Nature de la valeur {0,1} : entier, réel, caractère ou boolean
- Une valeur ou plusieurs valeurs à la fois
- Possibilité d'organiser les données pour faciliter la résolution.

Data

"Comment Organiser au mieux l'Information dans un Programme ?"

Structures de données

Tableaux

Enregistrement

Constante

Matrice

Scalaire

Donnée = Valeur(s)

Constante ≠ Variable

Type à définir
→ Définition + Déclaration

Type prédéfini
→ Déclaration

Une valeur → scalaire → 1 case

Plusieurs valeurs → Besoin d'organisation

Valeurs homogènes / hétérogène

Vecteur (Tableau)
→ n cases de type X

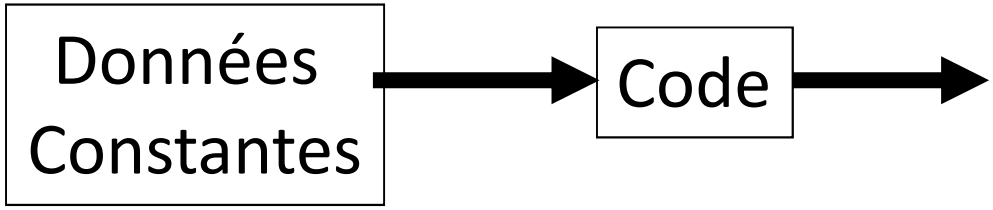
Matrice (tableau de tableaux)
n*m cases de type X

Enregistrement → scalaire composé

Liste

cas particuliers : Pile + File

Les Données



- Pré-compilation
- Commande préprocesseur
- # define PI ...

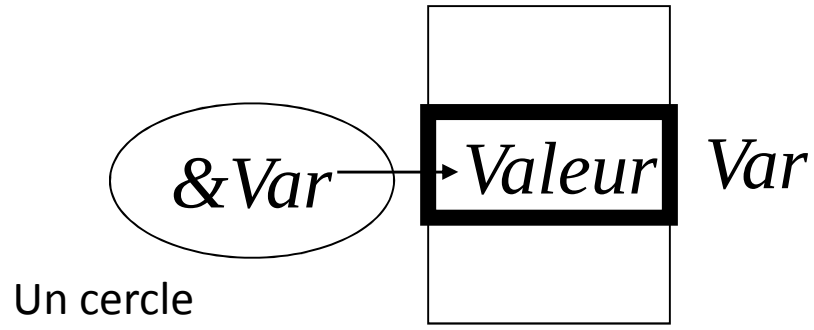
Variable :

- un Nom
- une Valeur
- une Case mémoire accessible par son Nom

= Nom → Utilisation de la valeur

Nom = → Modification de la valeur

&Nom → Où se trouve la case pour modification



Une donnée d'entrée ne doit pas figurée à gauche =

Type :

- Variable ou constante sont typées.
- Le type détermine les opérateurs que l'on peut lui appliquer.
- Le résultat d'une expression a un type.
- Dans une affectation la variable et l'expression doivent avoir le même type → Le type détermine les valeurs licites pour la variable et les opérations autorisées

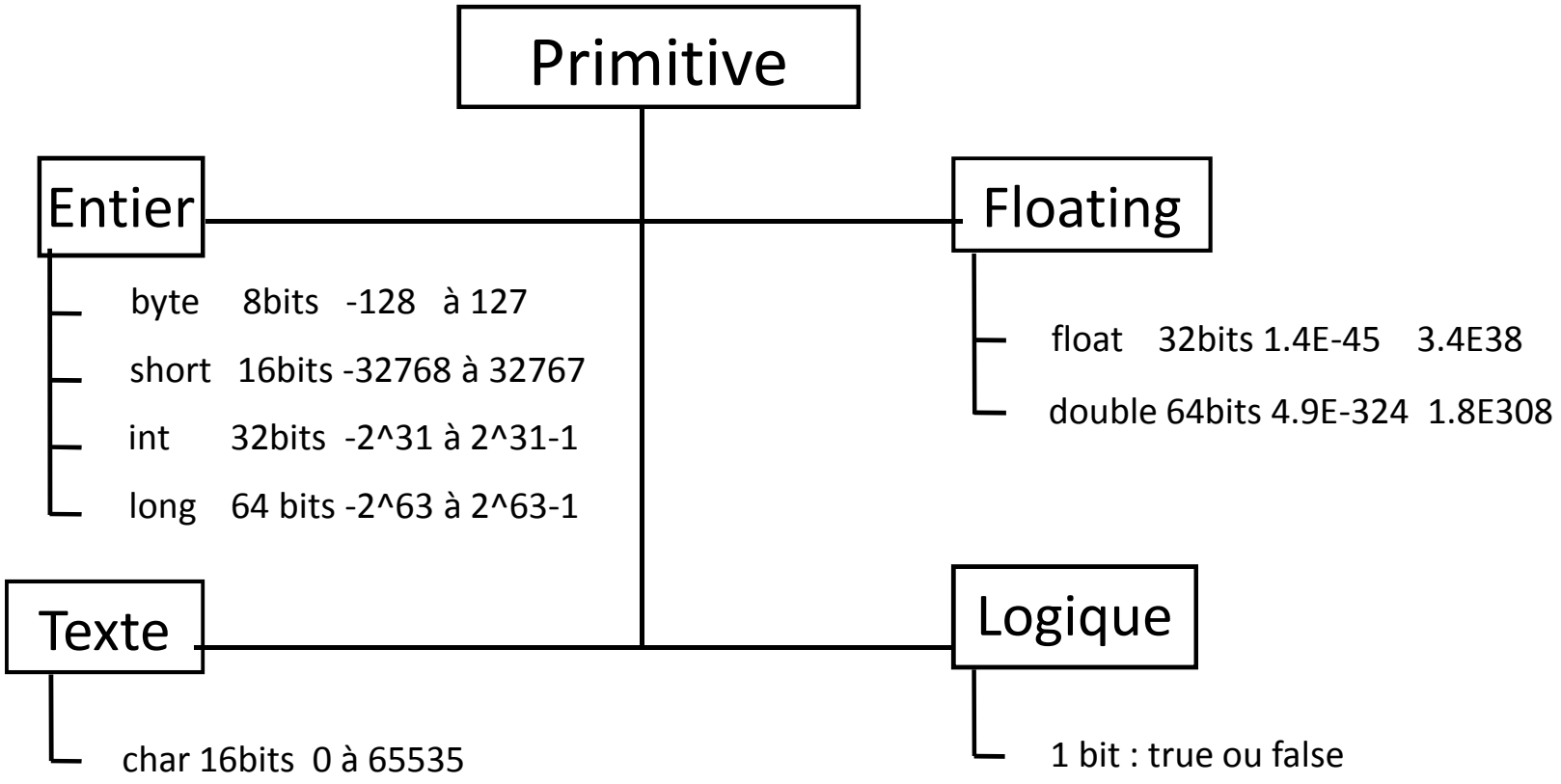
N vs. R → cast

Caractère vs. Entier (ASCII)→ Ordre

R/W des valeurs → Formatage

Ecran/Clavier	Programme
Caractères	Données typées

Types de base



La taille d'un type : ~~nombre~~ d'octets & capacité

Transformer un entier en flottant et inversement : le cast

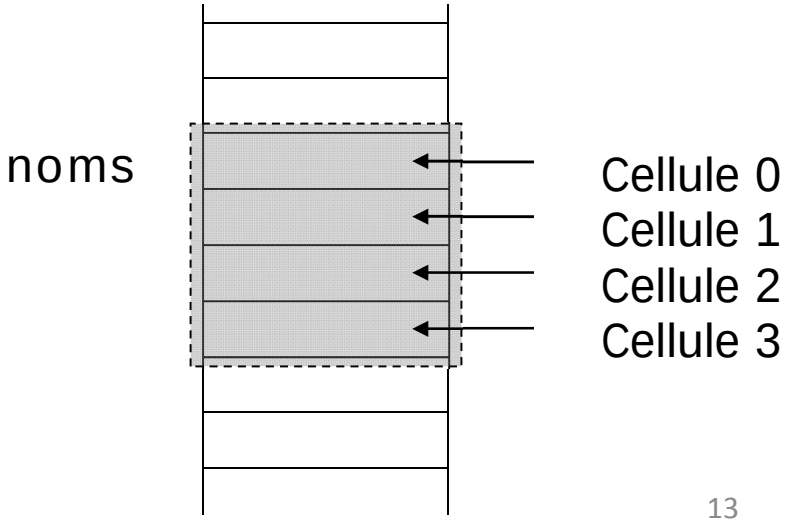
Donnée avec plusieurs valeurs à la fois → Organisation

Les tableaux

- tableau = suite séquentielle de cellules mémoire contenant des données de même type
- La taille d'un tableau doit être fixé dès sa déclaration
- Accès indexé (de 0 à n-1 pour un tableau de n éléments)

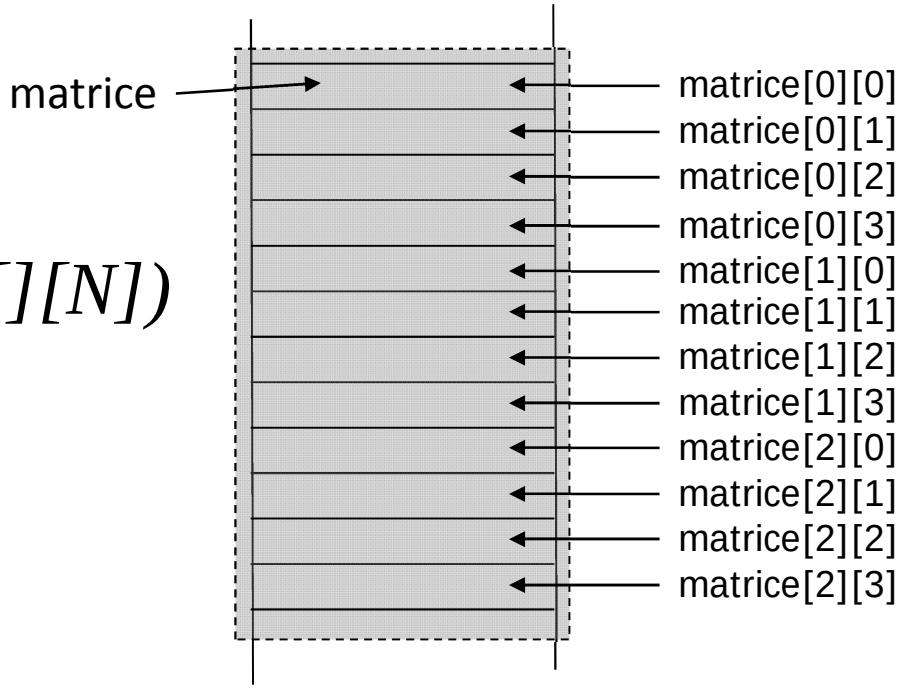
- ? Confondre l'indice et la valeur
- ? Jeu d'indices et pas
- ? Initialiser le tableau ou pas

Paramétrage d'un tableau passé



Représentation matrice

Passage de matrice : *F(type M[][N])*



Représentation pratique

	0	1	2	3
0	matrice[0][0]	matrice[0][1]	matrice[0][2]	matrice[0][3]
1	matrice[1][0]	matrice[1][1]	matrice[1][2]	matrice[1][3]
2	matrice[2][0]	matrice[2][1]	matrice[2][2]	matrice[2][3]

Pointeur de pointeur

Chaîne de caractères

Tableau de caractères : taille fixe
Chaîne de caractères = Tableau de caractères + '\0' :
taille à déterminer

Remarque :

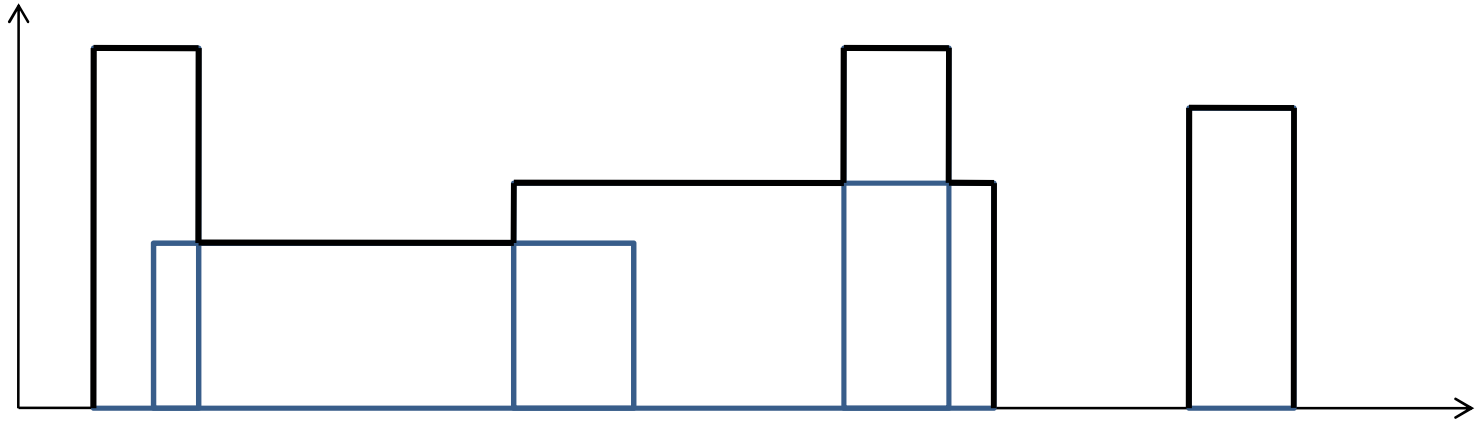
- Eviter d'utiliser les fonctions prédéfinies
- Les fonctions classiques: copie, comparaison, ordre, longueur
- Lecture et Ecriture de chaîne sans et avec espaces.
- Dans l'énoncé de pb : il faut préciser les fonctions prédéfinies qui peuvent être utiliser.
- Une fonction développée peut être réutiliser.

Enregistrement

Des données du problème hétérogènes dont la déclaration séparées peut nuire à la résolution du problème

Exemples :

- Les données d'un élève triées selon un critère
- Un rectangle dans le plan est soit :
  - 4 coordonnées x1, x2, y1, y2 (réelles)
  - 4 points p1, p2, p3, p4
  - 2 points phg, pbd
  - 3 valeurs x1, x2, h si le rectangle (cas du pb Horizon)



Fonction retournant le milieu d'un segment

Cellule d'une liste

Membre pointeur pour allocation dynamique (en profondeur?)

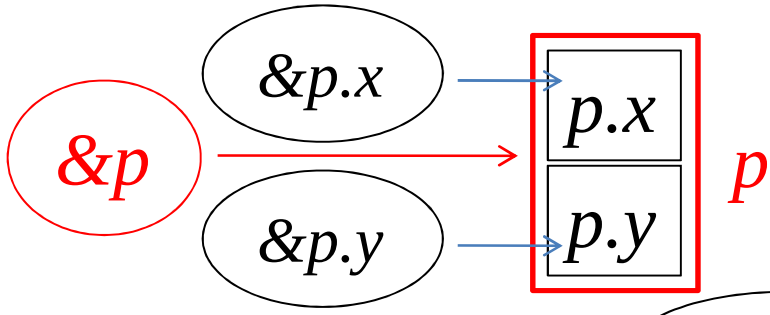
Affectation totale ou partielle

typedef

Accès à travers un pointeur (*). Plus tard →

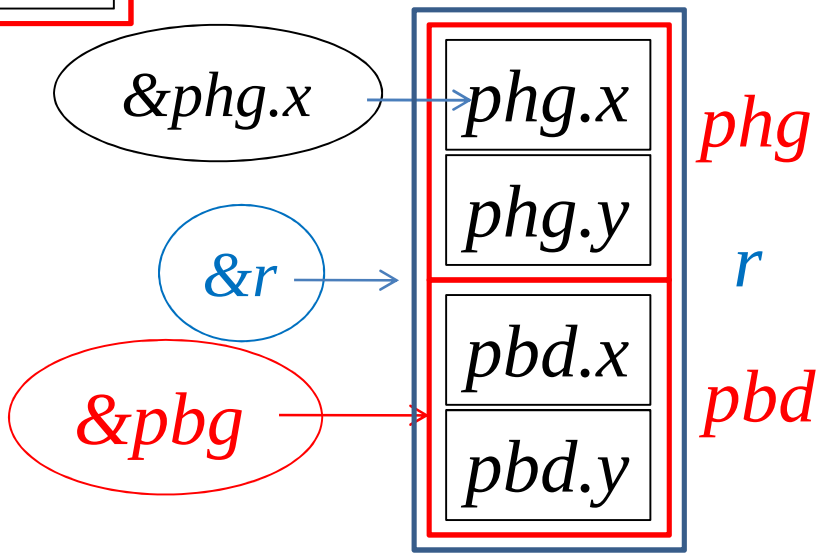
```
struct Point{
int x, y; };

struct Point p;
```



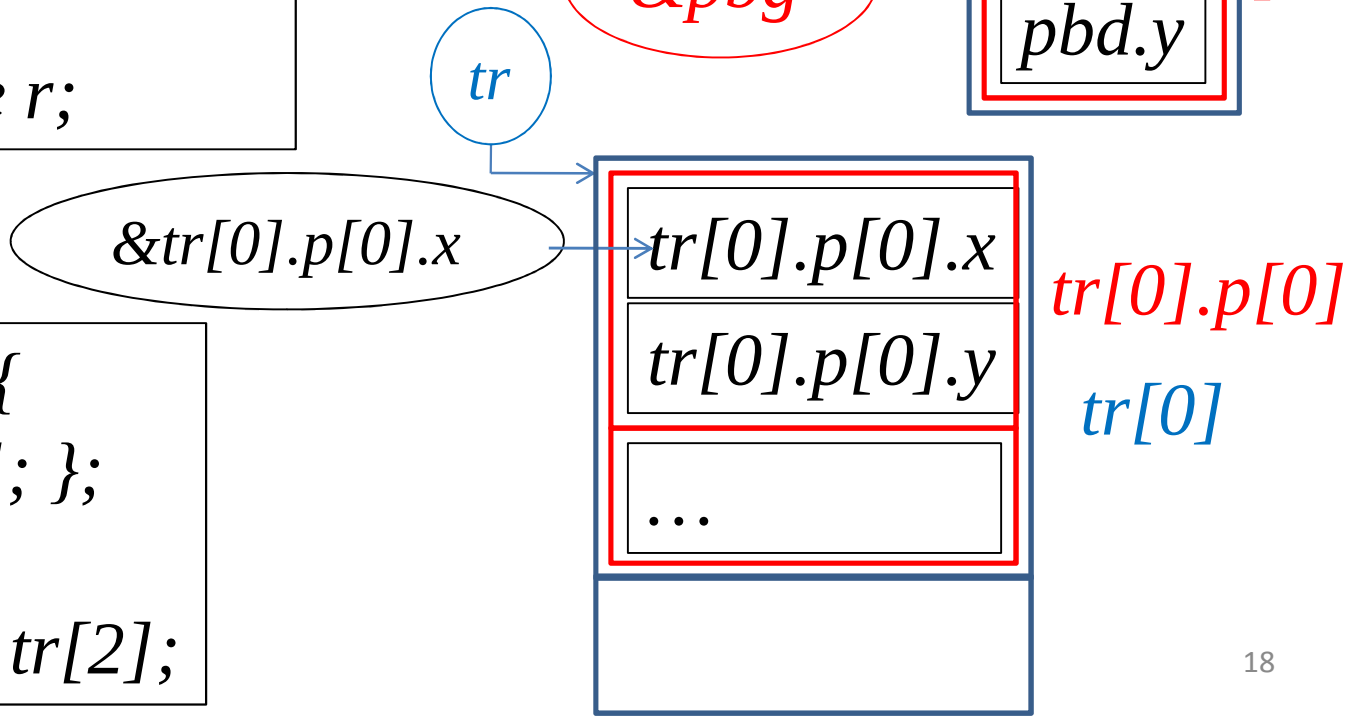

```
struct Rectangle{
struct Point phg, pbg; };

struct Rectangle r;
```




```
struct Rectangle{
struct Point p[2]; };

struct Rectangle tr[2];
```



Primitives d'algo → actions possibles

Modification de données → affectation

E/S → Affichage texte et donnée + initialisation

Sélection d'instructions & condition et !condition

Condition simple / composée

Répétition d'instructions

Décomposition et analyse descendante

Décomposition et fonctions

Récursivité & récursivité/Itération

Composantes d'un algorithme

- Les données
  - intrants
  - extrants
  - données intermédiaires
- Les opérations
  - saisie de données d’entrées
  - ordonnancement des actions
  - présentation des résultats

Comment présenter un algorithme de façon structurée

- On doit retrouver les 3 éléments suivants :
  - *Description de ce que fait l'algorithme :* Texte
  - *Constantes et Variables utilisées*
  - *Logique de l'algorithme*

    Description détaillée des opérations réalisées par l'algo

Un programme est une suite d'instructions :
- E/S
- Affectation
- Affectations sous condition (Expression logique = 0 ou 1)
- Répétition d'Instructions (déterministe ou non déterministe)

Expression est soit :
- Arithmétique
- Logique (comparaison)
- Transfert

Opérateurs obéissent à des règles de priorité “imposé les ()”

Primitives fondamentales d'algorithmique
La séquence , l’affectation, l’instruction

Séquence:

- Les opérations d'un algorithmes sont exécutées en séquence
- L'ordre est important
- On ne peut pas arbitrairement changer cette séquence

Affectation :

- l’instruction principale
- Rôle : changer l’état des données

Sélection

La sélection permet de choisir des alternatives d'exécution selon des conditions sur les données

```
si (condition)
....
```

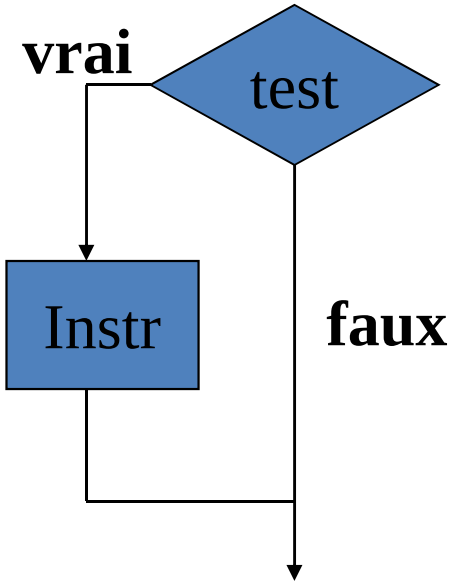
```
selon()
cas :
.
.
sinon :
```

```
si (condition)
....
sinon
....
```

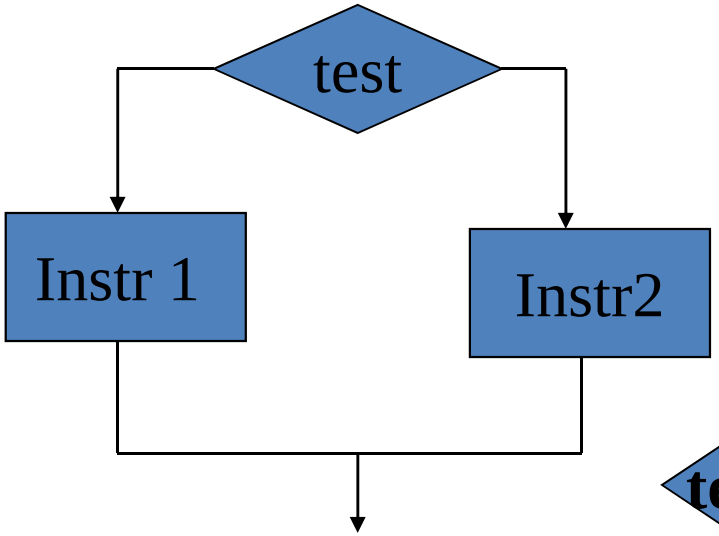

```
si (condition)
....
si (!condition)
....
```

Types d'alternatives

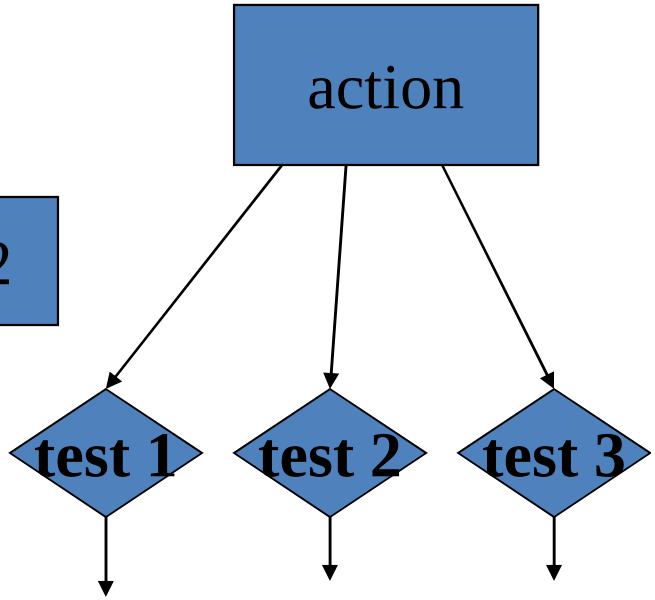
à une branche



à deux branches



à plusieurs branches



Opérateurs Arithmétique

- Le résultat peut dépendre du contexte

```
25 / 3 -----> 8
25.0 / 3 -----> 8.33
25 / 3.0 -----> 8.33
25 % 3 -----> 1
```

Opérateur Relationnel : ==, …

Opérateur Conditionnel : && ; || ; !

Il est parfois nécessaire, de traiter un certain nombre de fois une suite d'opérations.

Une structure répétitive peut être employée afin d'éviter une écriture séquentielle lourde (voir impossible.

Tant-que :

- Effectue une instruction (ou un bloc) tant qu'une expression booléenne est évaluée à true
- Réévalue l'expression à chaque tour de boucle
- On peut sortir de la boucle avec break;

- Existence d'une Condition d'arrêt.

La boucle Pour → compteur + condition d'arrêt (Le pas d'inc.)

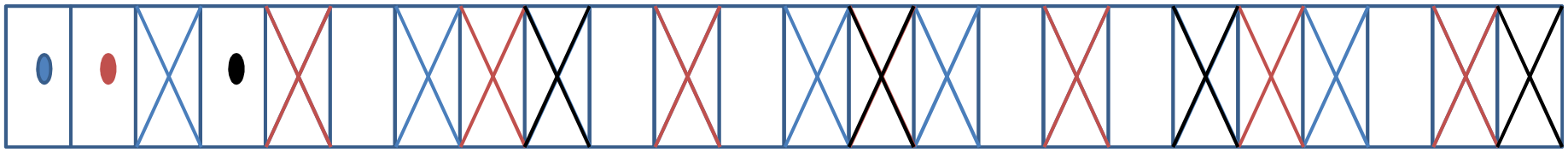
Equivalence avec tant-que et faire TantQue (jusqu'à?)

Parfois plus clair dans un programme

Sortie de la boucle prématurée

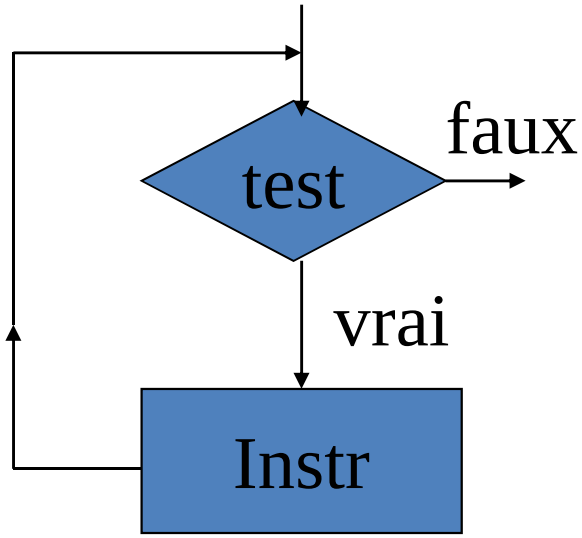
Regroupement et Eclatement de boucles

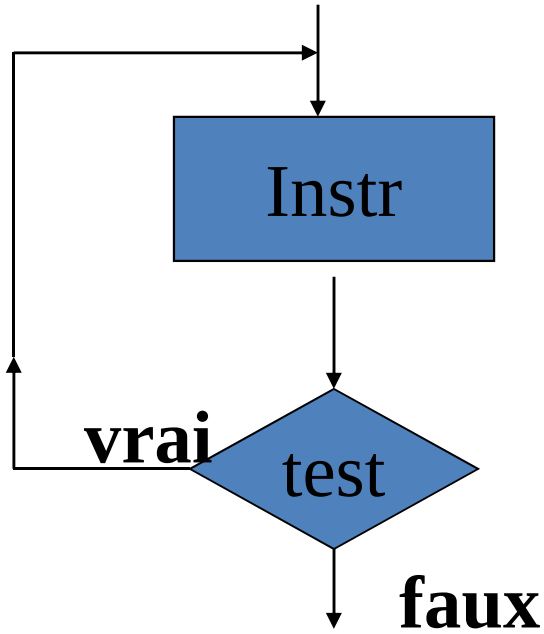
Exp. Crible d'Eratosthène



Vérification vs. Choix de valeurs (i non pair)

Représentations graphiques





TANT QUE
(le test AVANT Instructions)

Faire Instr TantQue
(le test APRES l'énoncé)

!test → inversion du faux et du vrai

Objectifs: maîtriser les compétences suivantes:

……..

Capacités visées:

……..

Prolongements, par exemple pour un en travail à la maison (Devoirs)

Conduire un raisonnement

Exemple de compétences: (activités en cours et activités en TP)

- Identifier les variables d'entrée, de sortie
- Utiliser une affectation
- Gérer des E/S (Saisir, Afficher)
- Prendre en main un environnement de programmation sur PC
- Découvrir progressivement la représentation des nombres dans la machine, maitriser l'affichage d'un résultat (curseur de l'écran)
- Utiliser la sélection
- Distinguer une égalité d'une affectation.
- Les problèmes de la gestion des égalités et des valeurs approchées dans la machine (précision)
- Etre capable de traduire la répétition d'une instruction.
- Le type de boucle utilisé
- Etre capable de gérer un compteur

….

Reprendre les compétences d'une manière Spiralée
en avançant dans le programme

Passage de l'algorithme au programme

- L'algorithme décrit la logique pour résoudre un probl.
- Le prog. implante l'algorithme dans un langage cible

- Éléments à considérer lors du passage de la structure d'algorithme au programme
  - Déclaration des variables et des constantes
  - Lecture des données (problème d'initialisation)
  - Affichage des résultats et mise en page
  - Contrôle des valeurs et cas particuliers

Mise en page (retour à la ligne)
Oublie de ;
Commentaires
Écrire associé à lire
Les { } forme un bloc, et cet ensemble est syntaxiquement équivalent à une instruction.
Déclaration dans le bloc ?

A un problème, il y a plusieurs solutions
(rester ouvert devant ces propositions)

Etapes de l'analyse descendante

-Diviser le problème en sous-problèmes de moindre difficulté

-Continuer la division jusqu'aux opérations élémentaires

-Marquer à chaque division la filiation

Exemples :

- Nombre d'Armstrong
- Calcul de l'exp(x)
- Nombres premiers
- Tri
- Formatage en C (transformation entier chaîne et inv.)
- Facteurs premiers (décomposition, récursivité, Itération)

…

Les fonctions et procédures

En mathématiques : Une relation qui associe les éléments de X aux éléments de Y

$$f : X \rightarrow Y$$

En programmation : Une partie d'un programme qui effectue des opérations à partir de données et qui renvoie un résultat.

- Paramètres changent le comportement de la fonction
- Résultat est typé
- Comportement stable (même X donne même Y)
- La fonction est à déclarer (ou l'ordre), à décrire et à appeler.
- Initialisation lors de l'appel
- Une seule valeur retournée (à cause de l'affectation)
- Absence de retour → procédure

- Intérêt :

– Décomposition du problème

- Présentation modulaire

– Réutilisation

– tests unitaires

- Faciliter la résolution et la correction (CNC)

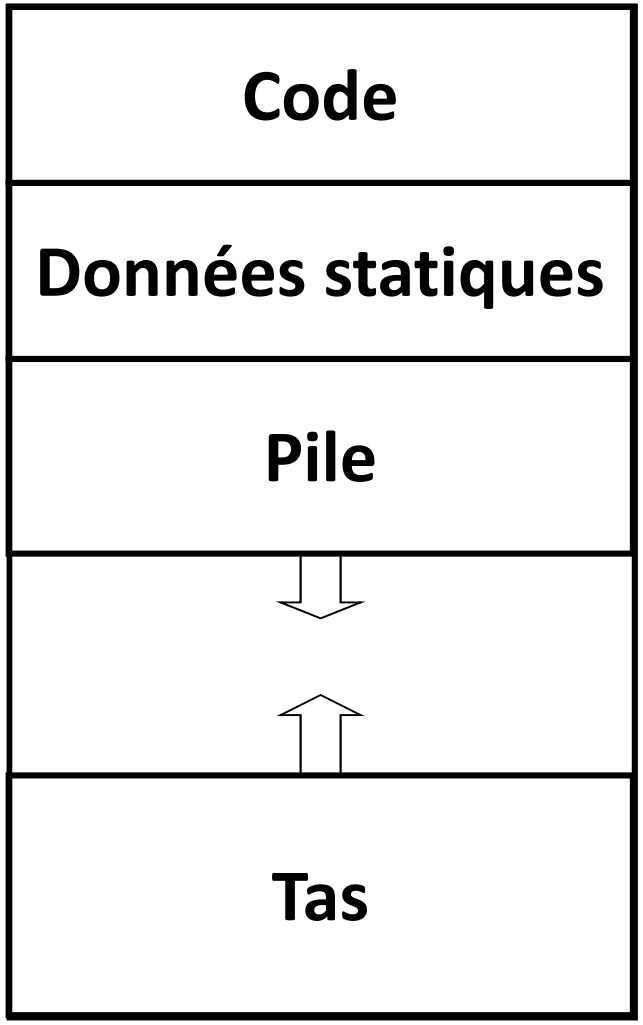
- ***Point de départ***

Déterminez les intrants, les extrants et les données intermédiaires dans le bloc de la fonction

- C n'autorise pas les fonctions imbriquées.

Fonction main

- Appelée en premier
- main appelle les autres fonctions nécessaires pour exécuter l'application
- Que se passe-t-il lors de l’exécution?



Code du programme

Valeurs constantes

Piles d’appels de fonctions

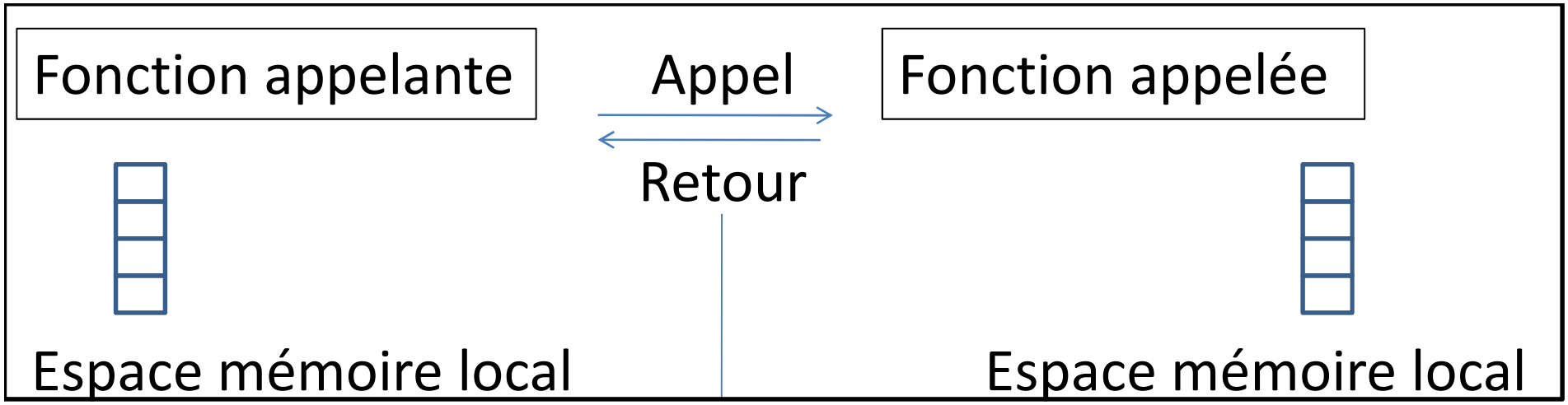
Allocation dynamique de mémoire

Données & fonction

- Passage de paramètres
- Valeur de retour
- Données globales
- Espace mémoire réservé dans la fonction

Durée de vie des variables

- Variable locale : déclarée à l'intérieur d'une fonction (ou d'un bloc ?). portée : la section de code où déclarée.
- Variable globale : variable déclarée en dehors des fonctions.
- Variables permanentes (ou statiques) : *static ?*.



Toujours passage d'une valeur :

valeur de la case ou valeur de l'adresse :

- Valeur de la case (copie)

  (besoin de la valeur au niveau de l'appelée)
- Valeur de l'adresse (modification dans l'appelée),

  utilisation de pointeur

  équivalence *p ≡ var (dans l'appelant)

  p ≡ &var (dans l'appelant)
- Valeur de retour (1 seule)

- Cas d’un tableau statique
- Cas d’un tableau dynamique (taille non déterministe)
- Besoin de création de zone chez l’appelant

- Tjrs de l’appelant vers l’appelée
- Calcule de la taille chez l’appelant (id. au cas statique)
- Chez l’appelée malloc (TAS ?)

Intérêts des pointeurs

- Gestion de l'espace mémoire en cours d'exécution (pas pour optimisation)

  tableau de taille indéterministe (utilisation de malloc)

Si : taille bornée → taille = borne

taille très grande

- Manipulation du tableau par l'écriture courante

- Modifications de variables passées en paramètres

Opérateur *

- Précise la déclaration d'un pointeur.
- Permet d'accéder au contenu d'une variable par pointeur :

$$type\ \ x, y, *p\ ;$$
$$x = 10\ ;$$
$$p = \&x\ ; \quad \rightarrow \quad p \equiv \&x$$
$$\rightarrow \quad *p \equiv x$$

- Fonctionne aussi avec les adresses :

$$*(tab + 3) \equiv tab[3]$$

~~Pointeurs de fonction~~

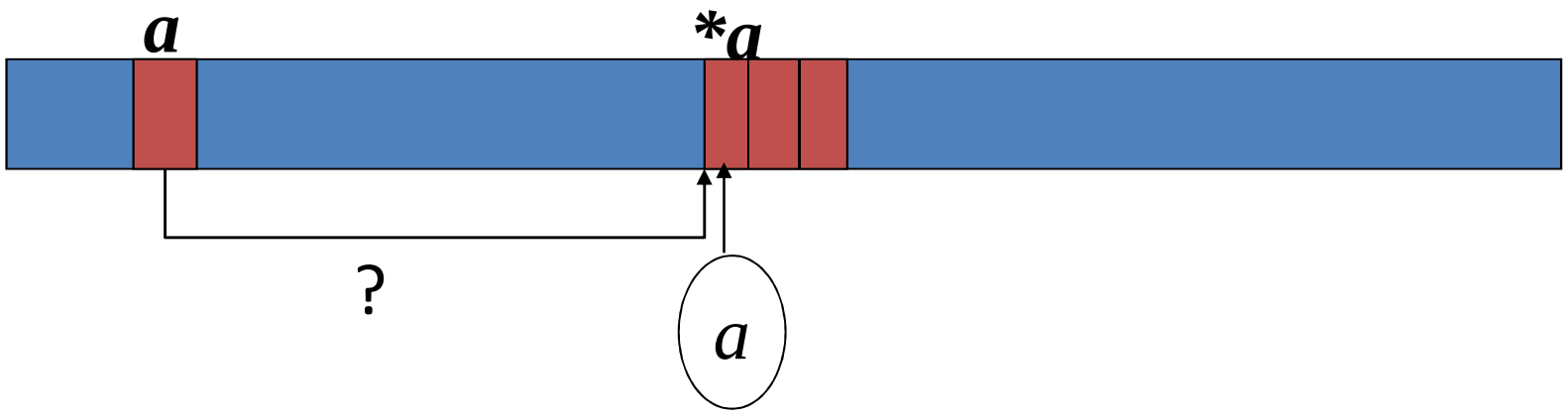
Type pointeur

Allocation dynamique et assignement

int a = malloc(3*sizeof(int));*

int a = (int*)malloc(3*sizeof(int));*

int a = NULL;*



calloc et realloc

Désallocation dynamique : *free(a);*?

Arithmétique d'adresses dans les tableaux

- Les adresses-mémoire du tableau sont contiguës

  $\Rightarrow$ on peut utiliser les opérations arithmétiques + et –

$$\Rightarrow \quad T \text{ ou } T+0 \quad \Leftrightarrow \quad \&(T[0])$$
$$T + 1 \quad \Leftrightarrow \quad \&(T[1])$$

Arithmétique de pointeurs ?

Structures de données linéaires

Tableaux : Taille fixe , Accès direct

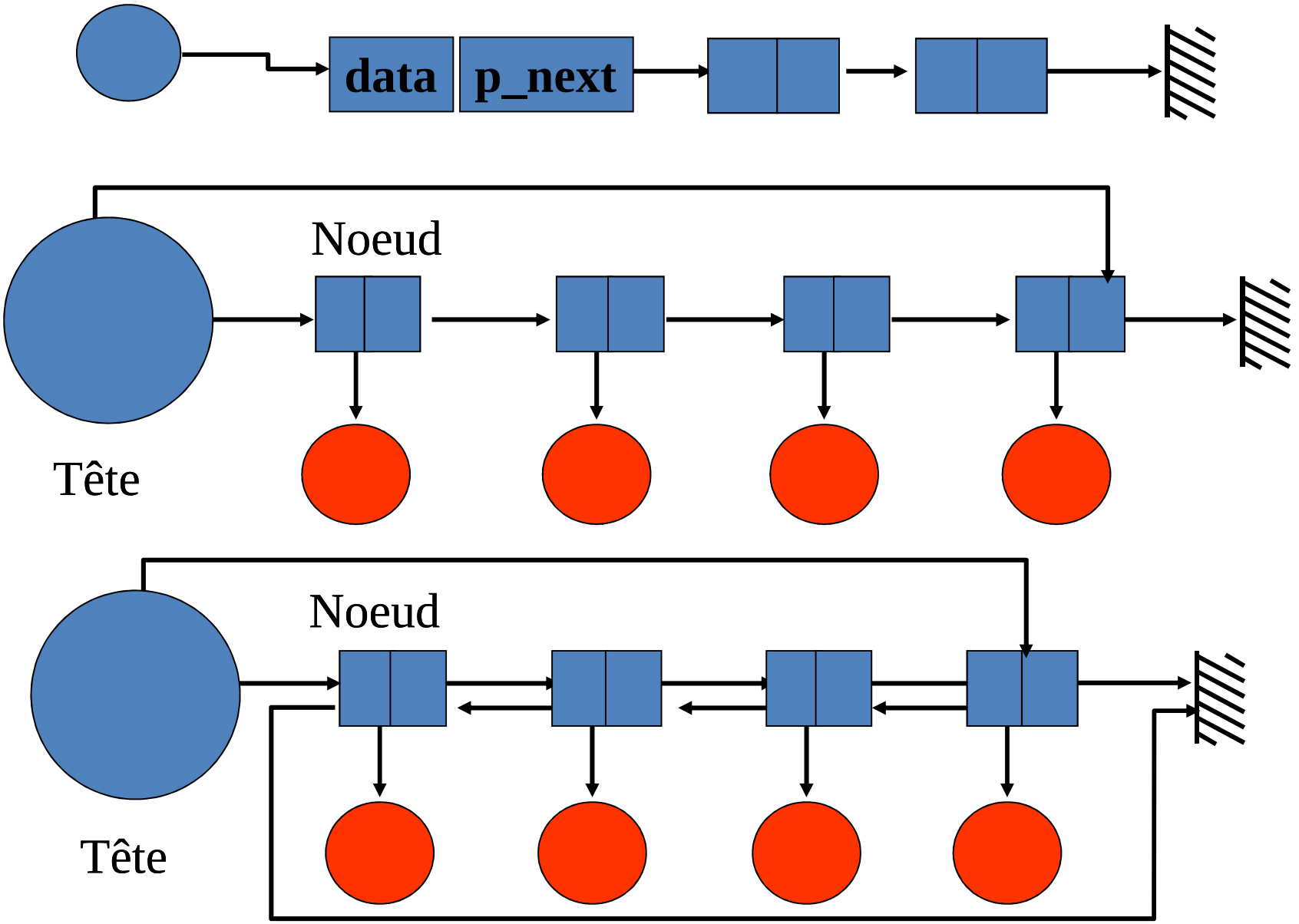
- Réajustement de taille coûteux en temps.
- Insertion d'élément onéreuse en temps.

.....

→ Intérêt des listes

Listes chaînées linéaire: Taille variable , Accès séquentiel

Liste chaînée : Structures



Liste chaînée : Implantation

- Tester la validité d'un pointeur avant de l'utiliser. (en TP)
- S'assurer de ne jamais perdre l'adresse d'une zone allouée dynamiquement.
- Désallocation par **free ?**

Remarque:
Pointeur de pointeur remplacé par retour de pointeur (modification de la tête)

Implémenter une structure en mémoire est une compétence recherchée mais parmi d'autres

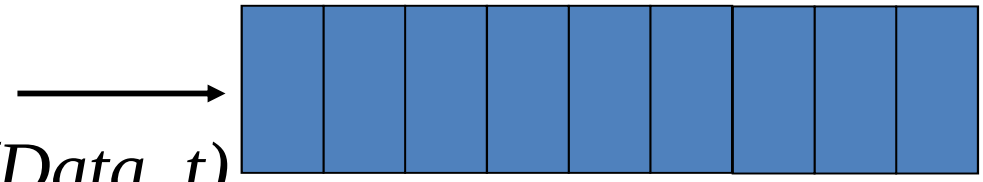
N'oublier pas de donner des structures prêtes à l'exploitation
Et de poser des problèmes sur ces structures

Liste chaînée : Spécialisations

Pile, ou Tas (Stack): structure LIFO

void push(Data_t) ↓ ↑ *Data_t pop(void)*

File, ou queue : structure FIFO



void push(Data_t)

Data_t pop(void)

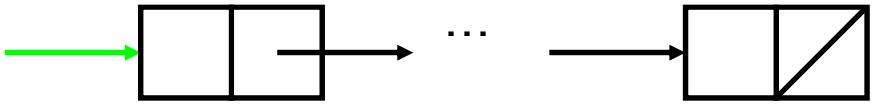
LIFO : listes et piles.

– ajout, lecture, suppression se font en tête de liste :

Insertion

Lecture

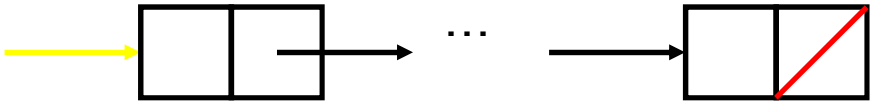
Suppression



FIFO: First In – First Out: File

– L'insertion se fait à la fin.

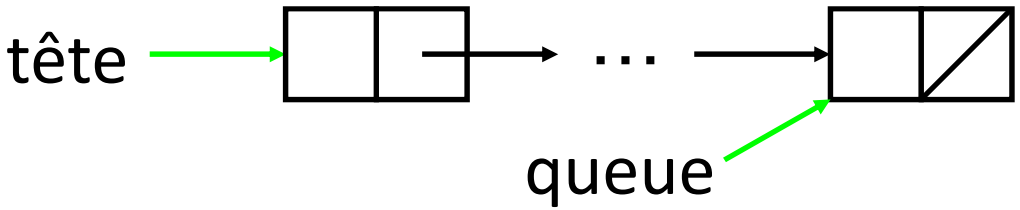
Lecture



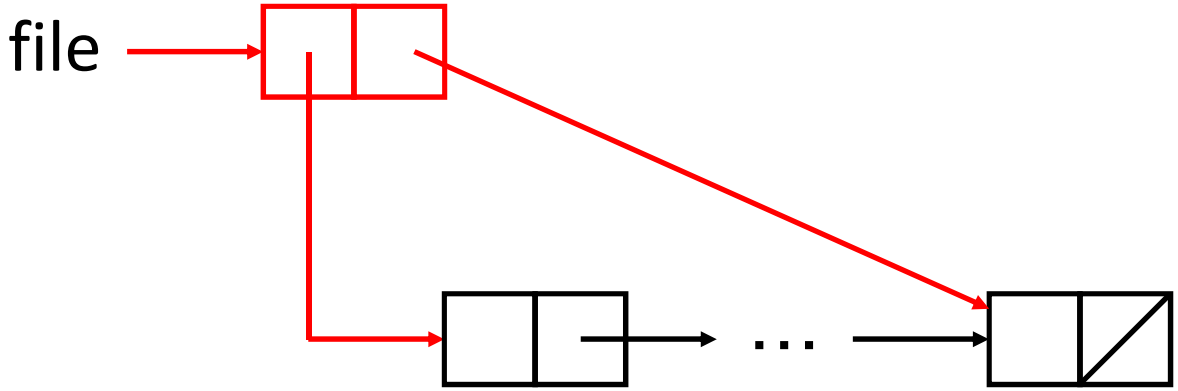
Suppression

Insertion

Souvent, on maintient deux pointeurs :



C’est plus pratique d’avoir une structure de synthèse :



Problème du choix de la tête
Curseur

Compétences au niveau des listes :

Savoir représenter une liste linéaire comme élément d'organisation de donnée capable d'optimiser un certain nombre de situation (nombre de données variable avec besoin de maintenance de la liste, …)

Vis-à-vis des élève :

Apprentissage des manipulation usuelles

Enrichir le chapitre par des cas d'utilisation avec une problématique.

La récursivité

- Méthode de programmation puissante qui permet d'exprimer d'une manière élégante la solution de problèmes difficiles
- Une fonction s'appelle elle-même (direct)
  - Une condition d'arrêt
  - Un appel
- Fonctionnement
  - La pile des appels
- Exemple
  - Les poupées russes
  - Tour d'Hanoï

Les questions clé de la récursivité

1. Comment exprimer la solution du problème en terme de la solution du même problème ?
2. En quoi chaque appel récursif diminue-t-il la taille du problème?
3. Quels cas du problème ont des solutions simples et serviront de cas de base?
4. Les appels récursifs convergent-ils vers un cas terminal ?
5. Itération & récurrence et parcours.
6. Dérécursivité (hors programme).

- **Récursivité simple**

  la fonction puissance $x \rightarrow x^n$.

$$x^k = \begin{cases} 1 & si\ k = 0 \\ x*x^{k-1} & sinon \\ ou\ 1/x*x^{k+1} & \end{cases}$$

Récursivité terminale ?

– **Récursivité multiple** : plus d'un appel récursif.

Calcul de combinaison

$$C_n^p = \begin{cases} 1 & si\ p=0\ ou\ p=n \\ C_{n-1}^p + C_{n-1}^{p-1} & sinon \end{cases}$$

– **Récursivité imbriquée**

$$A(m,n) = \begin{cases} n+1 & \text{si } m=0 \\ A(m-1, 1) & \text{si } m>0 \text{ et } n=0 \\ A(m-1, A(m, n-1)) & \text{sinon} \end{cases}$$

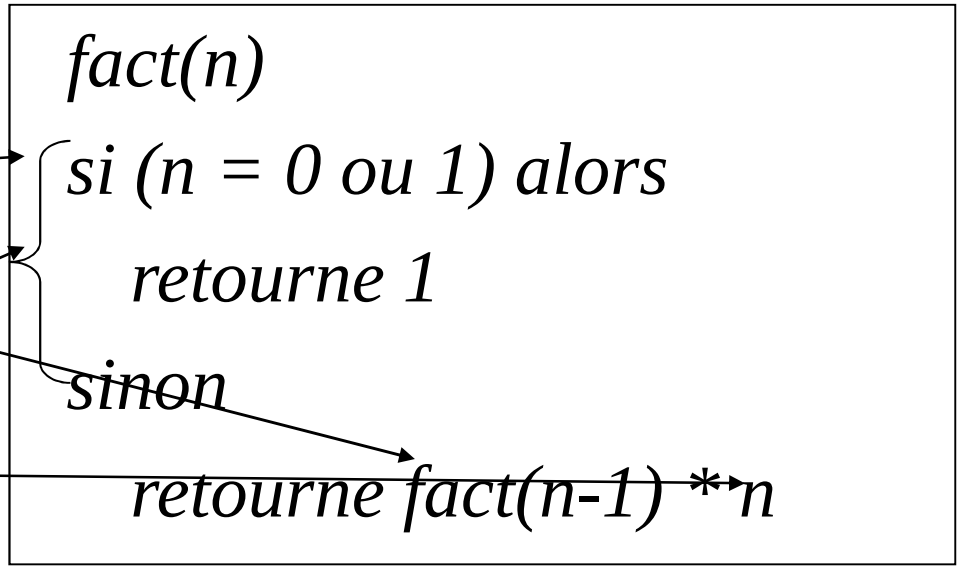
– **Récursivité mutuelle (hors programme)**

$$Pair(n) = \begin{cases} vrai & \text{si } n=0 \\ impair(n-1) & \text{sinon} \end{cases}$$

$$Impair(n) = \begin{cases} faux & \text{si } n=0 \\ pair(n-1) & \text{sinon} \end{cases}$$

Dissection d'une récursivité

- La condition d'arrêt
- L'appel récursif
- Le code avant l'appel
- Le code après l'appel
  - qui sera exécuté au retour



L'appel initial

Récursivité et boucle

- La récursivité peut simuler l'effet d'une boucle.
  - Algorithme récursif simulant une boucle
  - Reçoit en paramètre le nombre de boucles à réaliser
  - n : nombre de boucles à réaliser

```
void boucle(int n){
    if (n <= MAX){
            imprimer(n);
            boucle(n+1);
    }
}
```

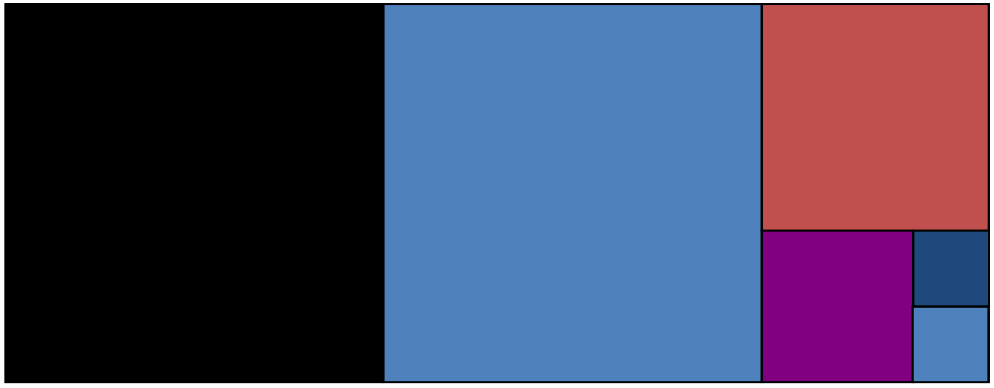
Qu'arriverait-il si on inversait ces 2 opérations?

Do while, while, for et récursivité

Le mécanisme de la récursivité

Algorithme d'Euclide pour le PGCD

- Trouver le plus grand carré permettant de le paver
- Découper en carré de côté = hauteur (largeur) du rectangle de façon à former un rectangle plus petit.
- Recommencer jusqu'à ce que la figure restante soit un carré
- Résultat : Le côté du carré est le pgcd de a et b.

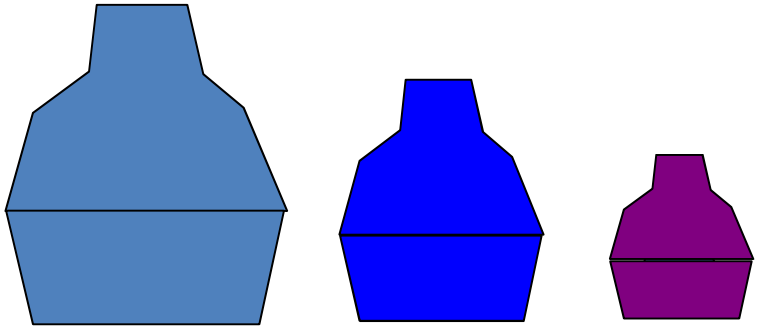


Rectangle
de cotés 65 et 25

```
while (a != b) {
if (a > b) a = a - b;
else b = b- a;}
```

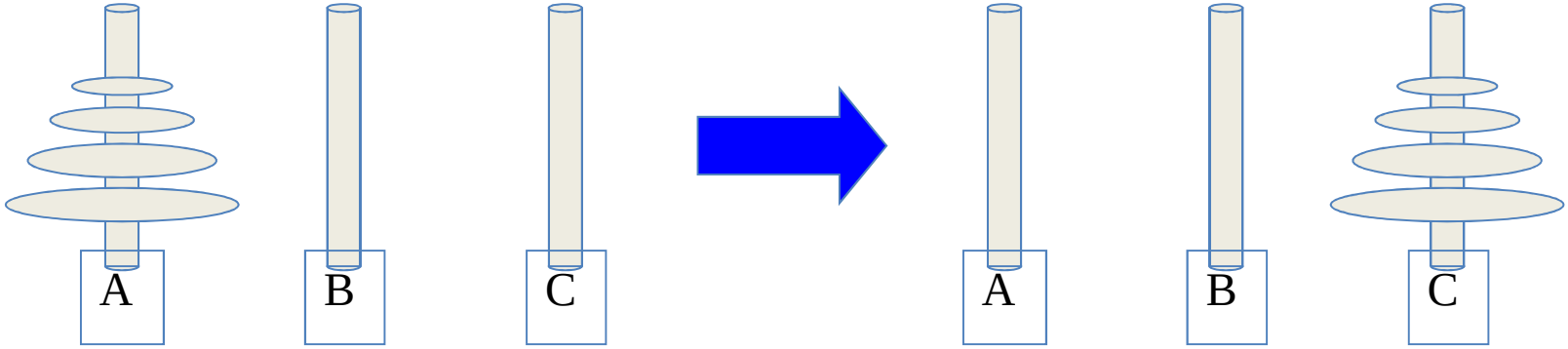
$$PGCD(a,b)=\begin{cases} PGCD(a-b,b) & si\ a>b \\ PGCD(a,b-a) & si\ a<b \\ a & sinon \end{cases}$$

Les poupées russes

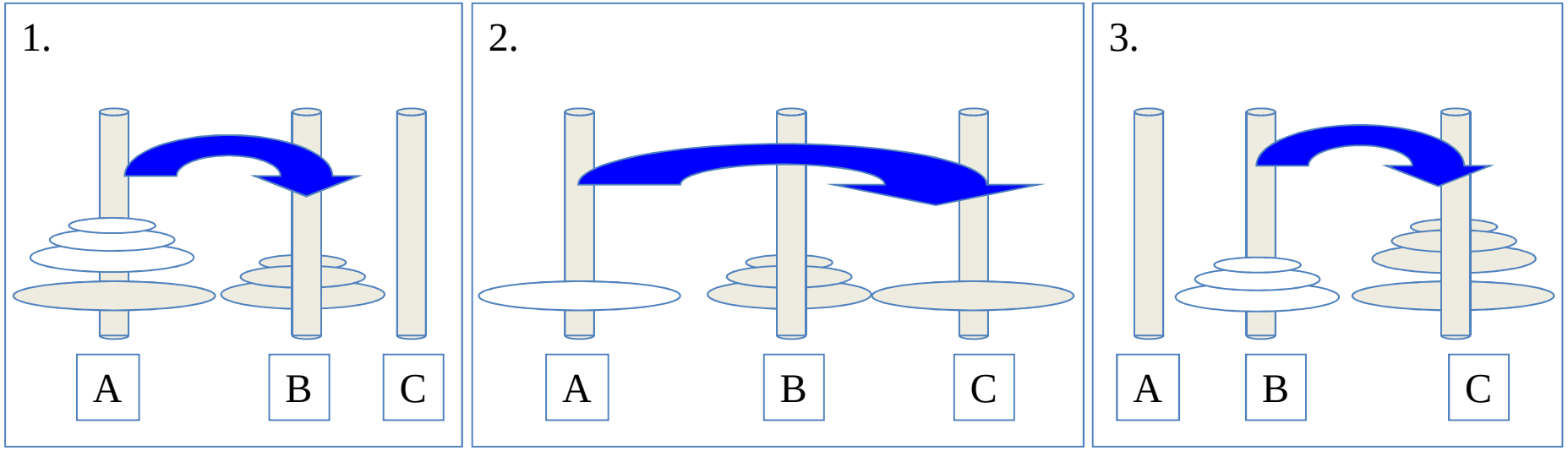


Les Tours de Hanoï

- Objectif : Déplacer tous les disques de la tige A à la tige C
  - La tige B sert d'emplacement temporaire
  - On ne peut empiler un disque sur un disque plus petit



- On décompose le problème en un problème plus simple
  1. On déplace les (n-1) disques, sauf le nème sur la tige B
  2. On déplace le nème disque sur la tige C
  3. On déplace tous les disques de la tige B sur la tige C



La pile des appels

- Une pile est utilisée pour mémoriser:
  - l'état des variables
  - L'endroit de retour
- À chaque appel, les valeurs des variables locales sont empilées
  - l'instruction `return` provoque la descente de la pile
  - cas d'une procédure ? (Exp. conversion)

Types de stratégies récursives

- Ascendante
- Descendante
- Par divisions successives (diviser pour régner)

Exp : Somme des carrés de x à y

$$Somme\ (x, y) = x^2 + (x+1)^2 + \ldots + y^2$$

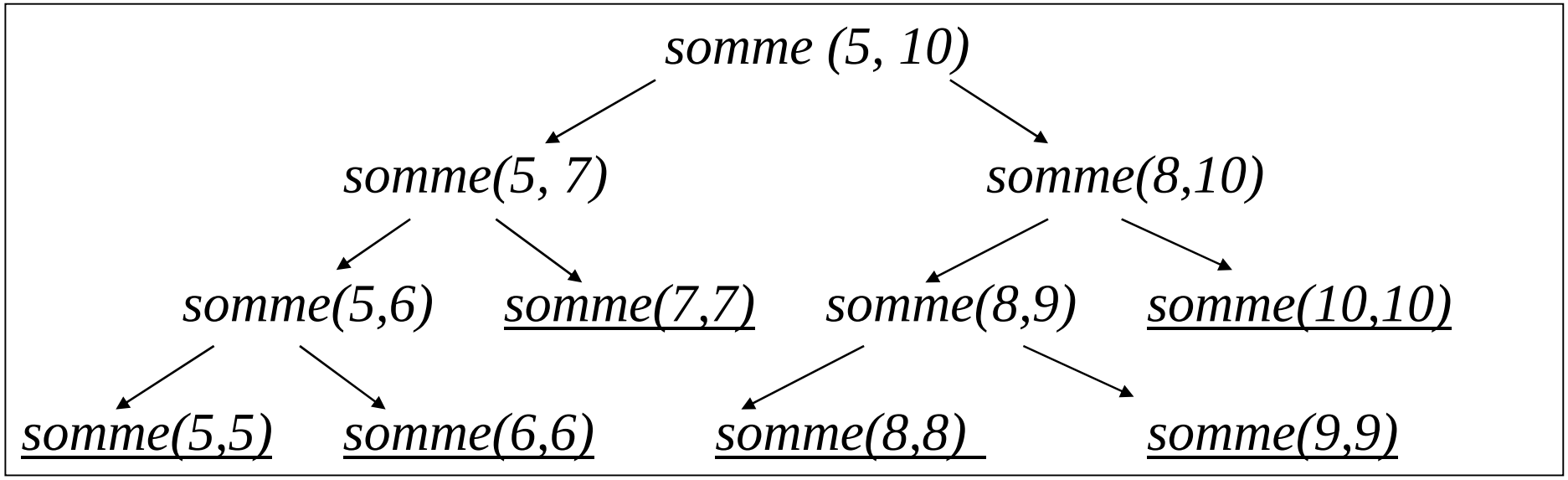
$$Somme\ (x, y) = \begin{cases} x^2 & si\ x = y \\ x^2 + Somme(x+1, y) & sinon \end{cases}$$

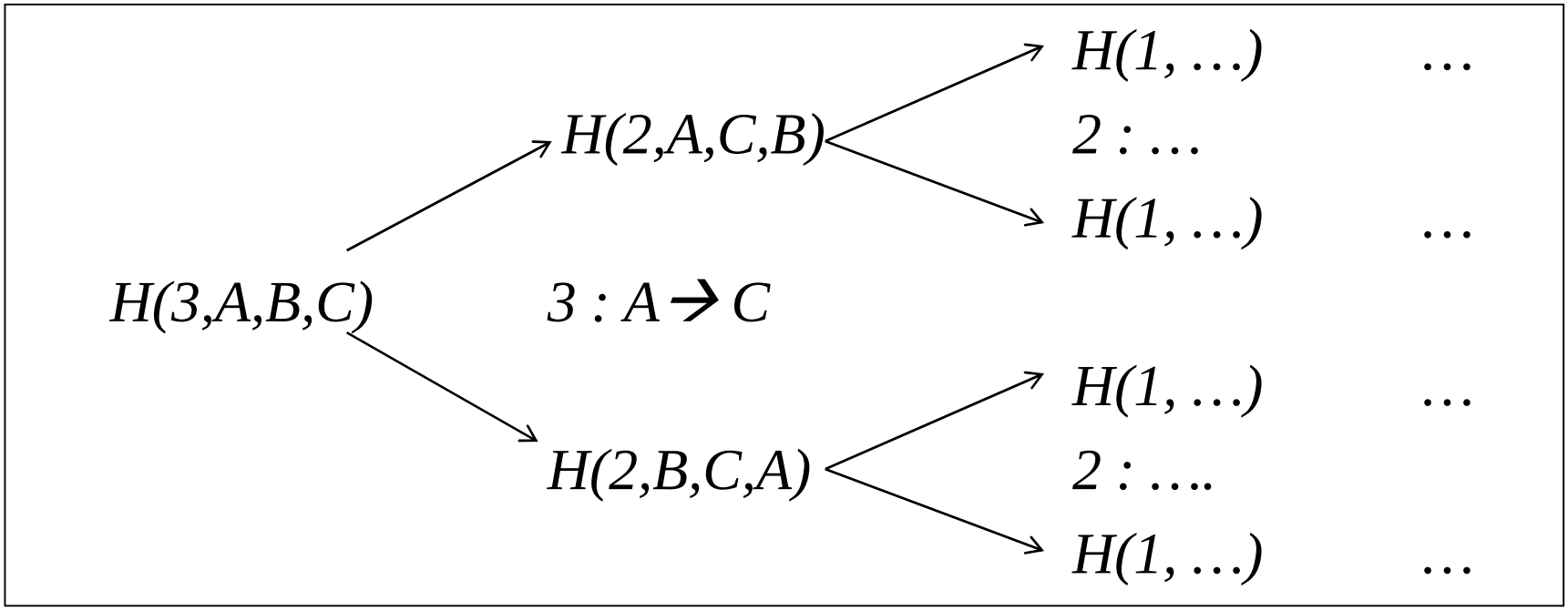
$$Somme\ (x, y) = \begin{cases} x^2 & si\ x = y \\ y^2 + Somme(x, y-1) & sinon \end{cases}$$

$$Somme(x, y) = \begin{cases} x^2 & si\ x = y \\ Somme(x, m) + Somme(m+1, y) & sinon \end{cases}$$

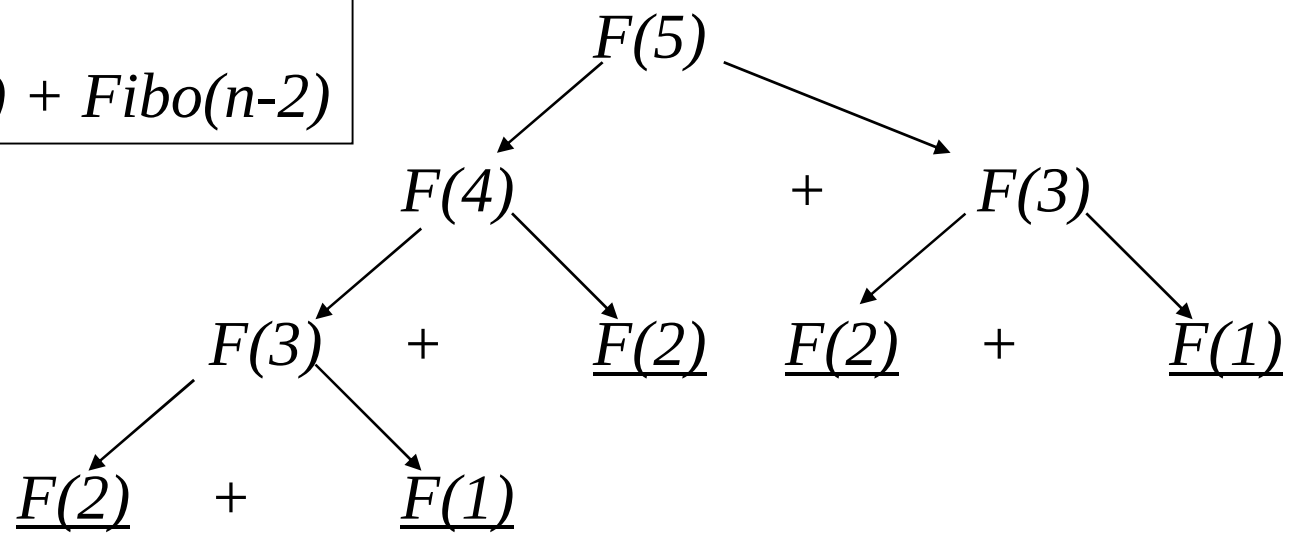
$$m = (x + y)/2$$

Arbre de l'algorithme





Fibo(1) = 1
Fibo(2) = 1
Fibo(n) = Fibo(n-1) + Fibo(n-2)

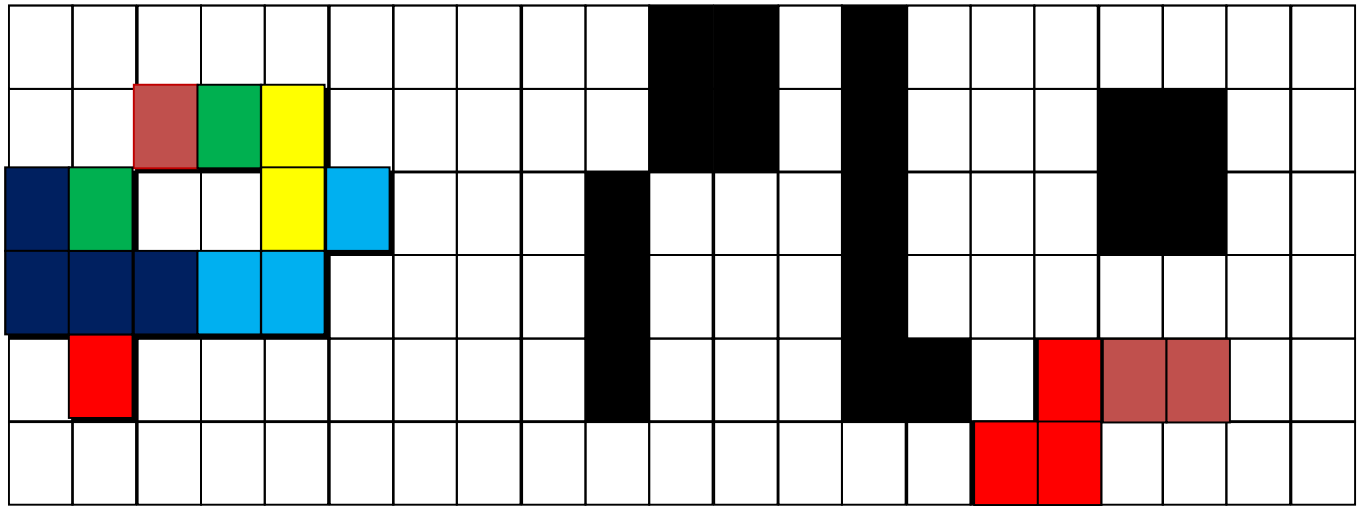


F(1)	F(2)	F(3)	F(4)	F(5)

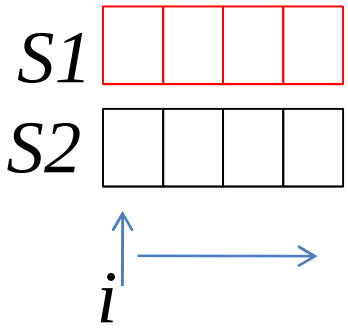
Taille du tableau ?
malloc

F(1)	F(2)	F(3)
F(2)	F(3)	F(4)

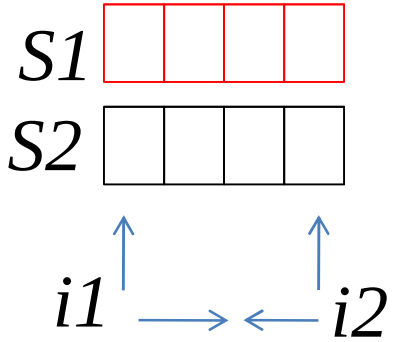
Coloriage d'objets



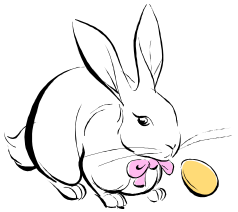
$$x^k = \begin{cases} 1 & si\ k = 0 \\ x * x^{k-1} & sinon\ k\ impair \\ x^{k/2} * x^{k/2} & si\ k\ pair \end{cases}$$

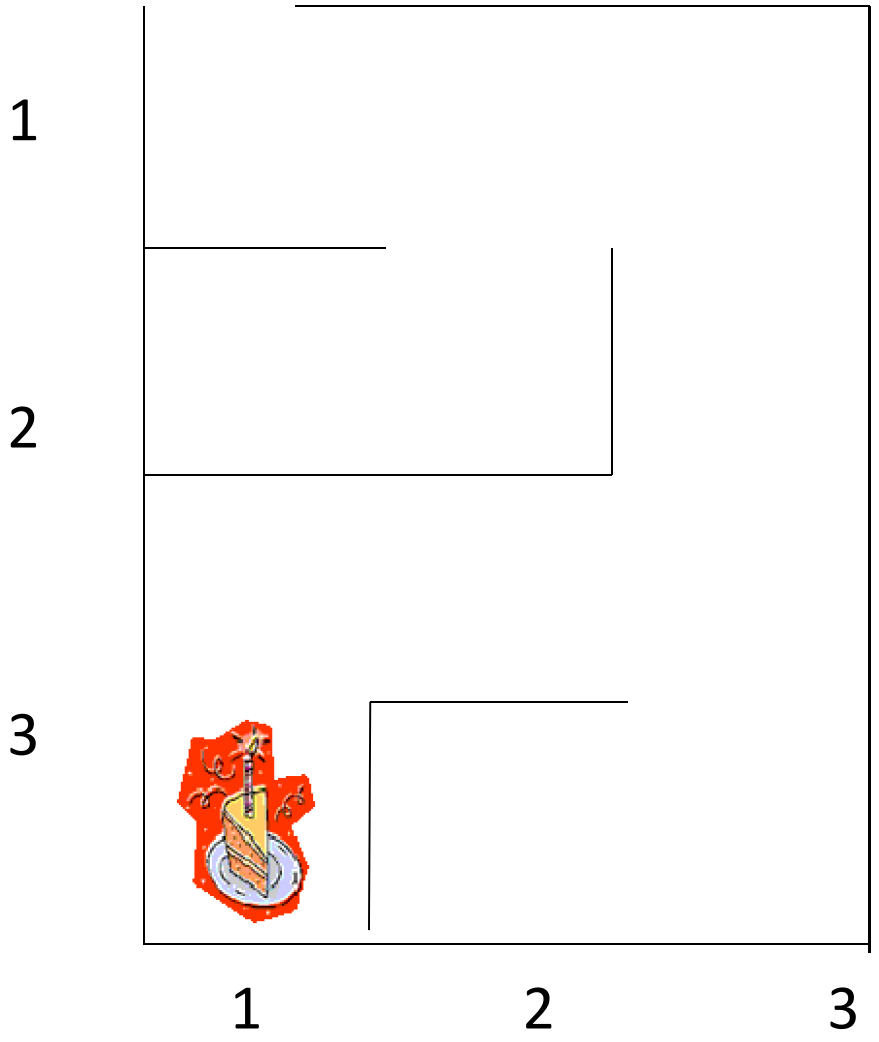


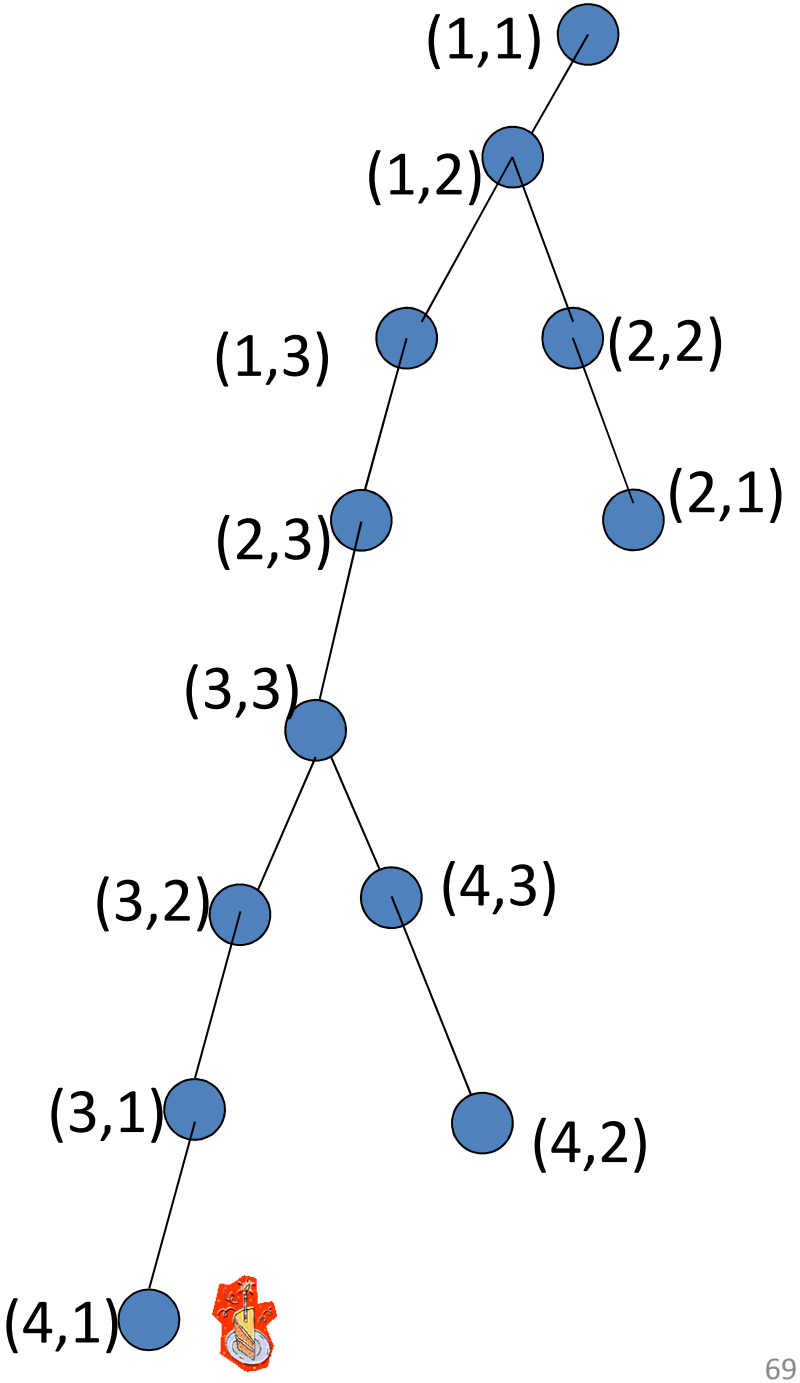
$$S1 ==? S2 \begin{cases} 0 & \text{si } l1 != l2 \\ 1 & \text{sinon si } i = l1 \\ 0 & \text{si } S1i != S2i \\ S1 =? S2 & \text{sinon avec } i=i+1 \end{cases}$$



Labyrinthe

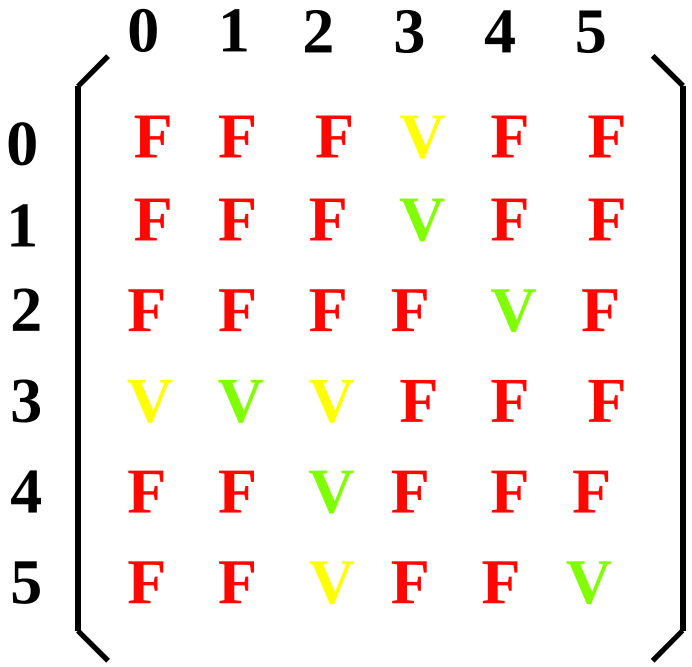


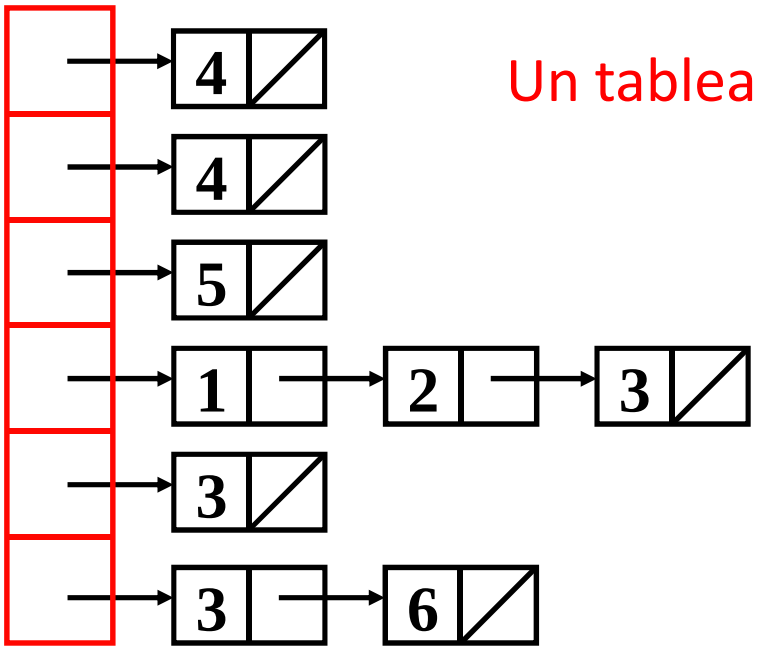




Algorithme retour sur trace « Backtracking »

- Le **retour sur trace** consiste à revenir légèrement en arrière sur des décisions prises afin de sortir d'un blocage. La méthode des essais et erreurs « trial and error » constitue un exemple simple de *backtracking*.





Un tableau de listes

Algorithmique et programmation
évaluation des compétences

L'élève sera capable …

- d'**analyser** le fonctionnement ou le but d'un algorithme existant ;
- de **modifier** un algorithme existant pour obtenir un résultat précis ;
- de **créer** un algorithme en réponse à un problème donné.

Envisager une évaluation par compétences.

Les compétences visées

- Lire un problème
- Analyser les situations et identifier les données pertinentes (DE, DS, DAux) et les représentations adéquates pour la résolution
- Approcher le problème à travers un exemple ou expression mathématiques (ex. récurrence)
- **Mettre en adéquation** le type des données avec l'objectif imposé.
- comprendre et analyser un programme ou fonction existant
- modifier un algorithme pour obtenir un résultat particulier

- mettre au point une solution algo : comment écrire un algo en langage courant en respectant un code, identifier **les instructions**, boucles, les tests, des opérations d'écriture, d'affichage... ;
- valider la solution par des traces d'exécution et des jeux d'essais
- adapter l'algo aux contraintes du langage : identifier si nécessaire la nature des variables... ; ?
- valider un programme simple.

➢ **Maîtriser** les différentes structures algorithmiques, leurs contraintes d'utilisation et leurs limites.

D'autres Compétences en jeu :

Exploitation des Notions du programme Maths, …

Culture générale : compression, cryptage, codage, changement de base, …

Raisonnement: Crible d'Eratosthène, Coloriage d'objets, …

….

La liste de compétences relative à l'algorithmique doit être enrichie dans les niveaux inférieur (connaissances) et supérieur (proposition de solution).

Pouvons-nous proposer des exercices pour chacune de ces compétences ?

- Comment concevoir des situations qui soient intéressantes, d'un point de vue mathématique et accessibles d'un point de vue algorithmique ?

Attention aux problèmes avec des résolutions mathématiques réduites.

ex. MxN, triangle de Pascal, et autres (se concerter avec les profs Maths)

- Peut-on évaluer isolément les compétences relatives à l'algorithmique ?

Exemple :

Expliquer ce que fait un algorithme

Modifier un algorithme afin qu'il réalise des objectifs

Compléter un algorithme

problème ouvert

Déterminer tous les nombres entiers naturels égaux à la somme des cubes de leurs chiffres.

$$1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$$

problème guidé

problème avec et sans exemple

Problème guidé + réutilisation

Le choix (ou non) d'intégrer les cas particuliers au début de l'analyse du problème ?

Problème avec le maximum de notions vues
Reprendre des épreuves existantes (Attention aux erreurs et adapter le texte)
Docs sur Internet ?
Méthode de résolution (rec vs. Iter) (choix évident ou pas)

Compétences évaluées	**I**ndicateurs	**Niveaux de maîtrise et autonomie**
Analyser la situation étudiée (langage naturel)	• Proposition d'une démarche de résolution • Ebauche d'une démarche algo en langage naturel	L'élève propose seul (**ou à l'aide des indices du prof**) une démarche cohérente de résolution

Mettre au point une solution algorithmique *Connaître ce qu'est un algo, la démarche algo et les énoncés nécessaires à sa représentation* (pseudo code)	**Détermination de la structure de l'algo** • Identification des DE, DS et DAux. • Détermination d'une séquence logique d' opérations. • Détermination des types de traitement appropriés à chacune des ops. (séquentiel, conditionnel, itératif) **Mise en forme d'Algo selon** des règles de syntaxe (pseudo code)	L'élève trouve seul (ou …) les DE, DS, DAux, la structure du traitement et validité de son écriture

Adapter un algo aux contraintes du langage choisi. (langage de programmation)	**Mise en forme en langage** • Utilisation de fonctionnalités du langage • Application des règles de syntaxe du langage utilisé.	L'élève identifie le type des variables (sans souci de taille), traduit la séquence des opérations et les conditions d'exécution, .

Déterminer la validité d'un algorithme. *Comprendre et analyser un algorithme éventuellement préexistant* *Valider une solution algorithmique*	**Vérification de la pertinence de la solution** compte tenu de la situation initiale. • Traces d'exécution. • Détermination des erreurs et des lacunes de la solution et correction. • Vérification de la terminaison d'un traitement. • Proposition d'une amélioration : généralisation, meilleure approximation…	L'élève repère et corrige seul (ou…) les erreurs L'élève fait preuve d'initiative dans les extensions souhaitées

Présenter sa démarche, les résultats obtenus	**Mise en forme de la production** • clarté de l'interface utilisateur (entrées, sorties) • Compte rendu soigné et lisible du travail demandé • Présence de toute l'info nécessaire à l'interprétation de l'algo.	La production contient des demandes d'entrées nécessaires, des lignes de commentaires judicieuses pour la compréhension du fonctionnement du prog.

utilisation et réutilisation de fonctions prédéfinies et définies	**Appel de fonction et** • Compréhension du fonctionnement • Spécification des entrées et sortie • Appel • retour.	L'élève est capable seul (ou...) de choisir, selon sa décomposition, les fonctions facilitant la résolution du problème. Capacité de désigner les Données échangées (Appel, paramètres, valeur de retour)

Perspectives :

L'optimisation temporelle et spatiale est une notion fondamentale en programmation.

Cependant, le programme dans sa version actuelle ne s'intéresse pas à cet aspect.

Une évolution futur du programme vers l'étude de la complexité pour cadrer les choix de solutions (linéaire, quadratique, …) et optimisation.

N'oublier pas Maple & 5/2 & rédaction Maple&C