DUT SRC – IUT de Marne-la-Vallée
04/11/2011
INF120 - Algorithmique

Cours 3
Méthodologie pour l'algorithmique
Les tableaux

Philippe Gambette

Sources

- *Le livre de Java premier langage*, d'A. Tasso
- Cours INF120 de J.-G. Luque
- Cours de J. Henriet : http://julienhenriet.olympe-network.com/Algo.html
- http://xkcd.com, http://xkcd.free.fr

Plan du cours 3 – Méthodologie, tableaux

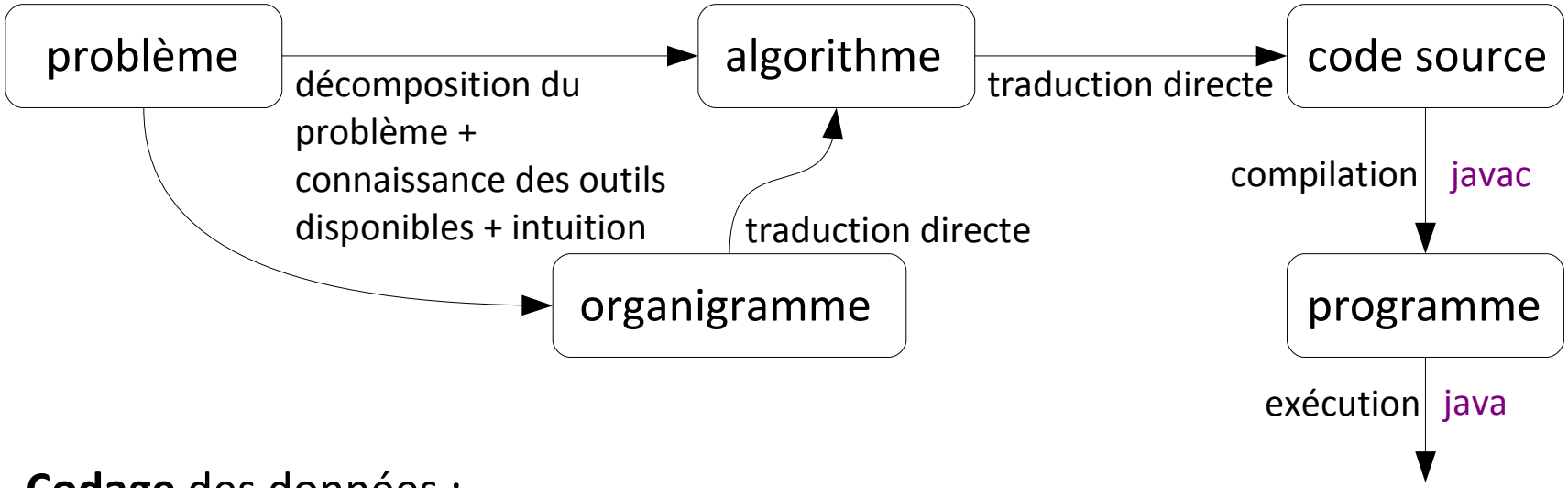
- Résumé des épisodes précédents
- Correction du QCM1 / présentation des DM1 & 2
- Compétences acquises en INF120
- Méthodologie pour l'algorithmique
- Exemples d'application de la méthodologie
- Les tableaux

Plan du cours 3 – Méthodologie, tableaux

- Résumé des épisodes précédents
- Correction du QCM1 / présentation des DM1 & 2
- Compétences acquises en INF120
- Méthodologie pour l'algorithmique
- Exemples d'application de la méthodologie
- Les tableaux

Résumé de l'épisode précédent

Du problème au programme pour le résoudre :



Codage des données :

• Pour chaque **type** de **variable** (entiers, flottants, chaînes de caractères, couleurs, booléens), une méthode de **codage** en binaire est choisie
(en Java : int, float, double, String, Color, boolean, ...)

• Définition d'**opérations de base** pour chaque type de données
(en Java : +, -, *, /, %, &&, ||, !, ...)

Plan du cours 3 – Méthodologie, tableaux

- Résumé des épisodes précédents
- Correction du QCM1 / présentation des DM1 & 2
- Compétences acquises en INF120
- Méthodologie pour l'algorithmique
- Exemples d'application de la méthodologie
- Les tableaux

Correction du QCM1

Voir corrigé distribué

Plan du cours 3 – Méthodologie, tableaux

- Résumé des épisodes précédents
- Correction du QCM1 / présentation des DM1 & 2
- Compétences acquises en INF120
- Méthodologie pour l'algorithmique
- Exemples d'application de la méthodologie
- Les tableaux

Compétences acquises en INF120

- Connaître les éléments de base d'un algorithme
- Savoir lire et comprendre un algorithme
- Savoir concevoir un algorithme pour résoudre un problème
- Savoir programmer un algorithme

Compétences acquises en INF120

Connaître les éléments de base d'un algorithme

Un algorithme résout un problème, en fournissant un **résultat en sortie** à partir de **données en entrée**.

Pour cela, il utilise plusieurs types d'**instructions** :
- des **affectations** dans des **variables** (mémoires)
- des **boucles**
- des **appels** à d'autres algorithmes
- des **tests**
- des “**lectures**” d'entrées et “**renvois**” de sorties

Chaque **variable** a un **nom**. On doit :
- la **déclarer** en définissant son **type** (ensemble de valeurs possibles)
- puis l'**initialiser** (lui donner une **valeur**)

avant de l'utiliser.

Compétences acquises en INF120

en pseudo-code

Connaître les éléments de base d'un algorithme

Un algorithme résout un problème, en fournissant un **résultat en sortie** à partir de **données en entrée**.

Entrées : entiers *i* et *j*
Sortie : somme *i+j* (entier)

une instruction par ligne !

Pour cela, il utilise plusieurs types d'**instructions** :

- des **affectations** dans des **variables** (mémoires) produit ← entier1+produit (variable / valeur)
- des **boucles** Tant que … faire : … Fin TantQue
- des **appels** à d'autres algorithmes Addition(3,5) — entrées de l'algorithme entre parenthèses, respectant l'ordre de déclaration des entrées
- des **tests** Si ... alors : ... sinon : … FinSi
- des "**lectures**" d'entrées et "**renvois**" de sorties

renvoyer *i+j*

Chaque **variable** a un **nom**. On doit :

- la **déclarer** en définissant son **type** (ensemble de valeurs possibles)
- puis l'**initialiser** (lui donner une **valeur** par une affectation)

avant de l'utiliser.

Variables : entiers *a* et *b* (type)

a ← 2
b ← *a*+2

Compétences acquises en INF120

en Java

Connaître les éléments de base d'un algorithme

Un algorithme résout un problème, en fournissant un **résultat en sortie** à partir de **données en entrée**.

public static int Addition(int i, int j){...}
type de sortie
entrées précédées de leur type

Pour cela, il utilise plusieurs types d'**instructions** :

instructions finies par ";"

- des **affectations** dans des **variables** (mémoires)
- des **boucles**
- des **appels** à d'autres algorithmes
- des **tests**
- des "**lectures**" d'entrées et "**renvois**" de sorties

produit = entier1+produit
variable
valeur

while(…){...}

Addition(3,5)
entrées de l'algorithme entre parenthèses, respectant l'ordre de déclaration des entrées

if(...){...}else{...}

return i+j;

Chaque **variable** a un **nom**. On doit :

- la **déclarer** en définissant son **type** (ensemble de valeurs possibles)
- puis l'**initialiser** (lui donner une **valeur** par une affectation)

avant de l'utiliser.

int a, b;
type

a = 2
b = a+2

Plan du cours 3 – Méthodologie, tableaux

- Résumé des épisodes précédents
- Correction du QCM1 / présentation des DM1 & 2
- Compétences acquises en INF120
- Méthodologie pour l'algorithmique
- Exemples d'application de la méthodologie
- Les tableaux

Méthodologie pour l'algorithmique

Savoir lire et comprendre un algorithme

Premiers éléments à identifier :
- qu'est-ce que l'algorithme **prend en entrée** ? **Combien** de variables, de quels **types** ?
- qu'est-ce que l'algorithme **renvoie en sortie** ? **Rien** ? Ou bien **une** variable ? De quel **type** ?

Ensuite :
- quels sont les autres **algorithmes appelés** par l'algorithme ?

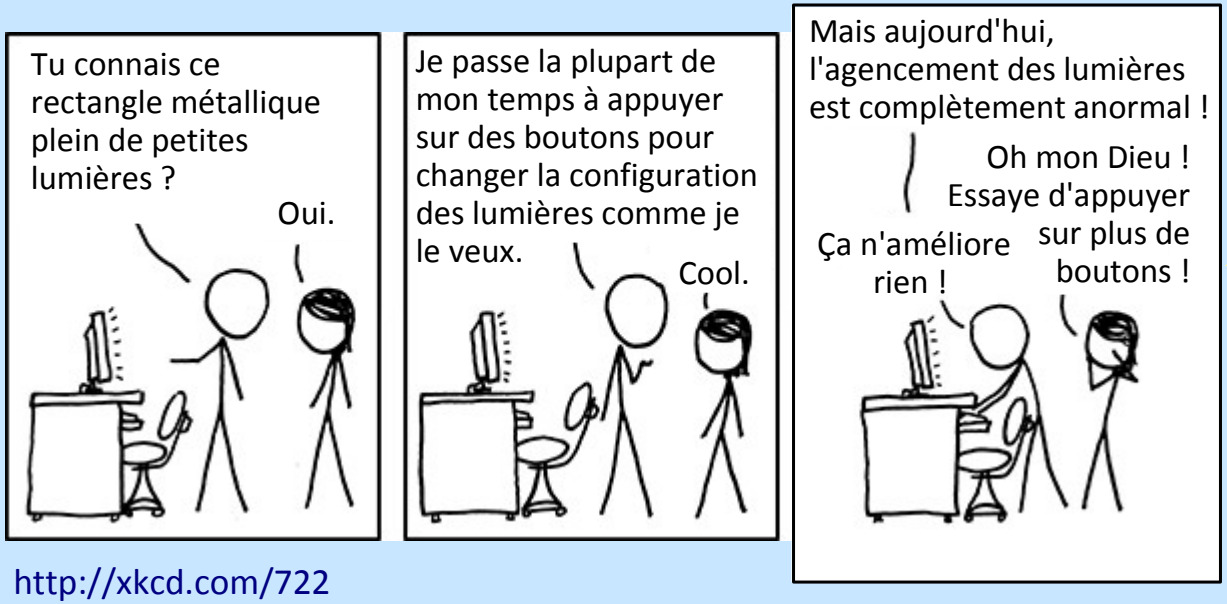
Enfin :
- faire la **trace** de l'algorithme, c'est-à-dire l'essayer sur un **exemple**
(… ou plusieurs pour passer au moins une fois par toutes les instructions de l'algorithme)
et voir ce que valent **toutes les variables à chaque étape** (et noter ces valeurs dans un tableau contenant une ligne par variable et une colonne par étape),
- noter en particulier le **résultat obtenu en sortie** pour une **entrée testée.**

Méthodologie pour l'algorithmique

Savoir concevoir un algorithme pour résoudre un problème

La "minute xkcd"

Problèmes informatiques



C'est comme ceci que j'explique mes problèmes d'ordinateur à mon chat. D'habitude, il a l'air plus content que moi.

http://xkcd.com/722
http://xkcd.free.fr?id=722

Méthodologie pour l'algorithmique

Savoir concevoir un algorithme pour résoudre un problème

Premiers éléments à identifier :
- quels sont les **outils à disposition ?** (pour ces outils : données en entrée, type de données en entrée, résultat en sortie, type de résultat en sortie, résultat attendu sur un exemple...)
- quel est le **comportement attendu** pour mon algorithme ? (données en entrée, type de données en entrée, résultat en sortie, type de résultat en sortie, résultat attendu sur un exemple...)

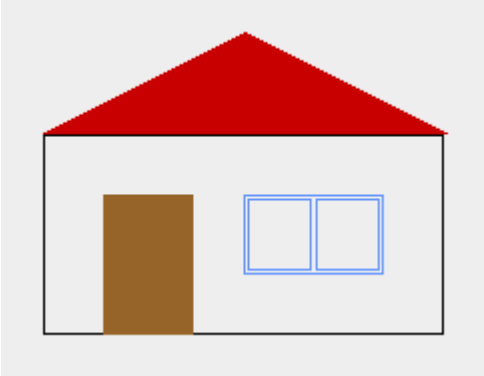
Ensuite, résoudre le problème en utilisant ces outils :
- comment résoudre le problème **étape par étape** ? (essayer sur l'exemple testé)
- est-ce que les **outils à disposition** sont **utilisables** pour réaliser chaque étape ?

Enfin :
- comment **structurer** l'utilisation des outils à disposition ? (**combinaison** des différents outils à l'intérieur de structure de **boucles**, de **tests**, utilisation d'un **organigramme**...)
- comment **décomposer** le problème ? (et **reformuler** chaque sous-problème pour le résoudre avec les outils à disposition, écrire un algorithme par sous-problème)

Plan du cours 3 – Méthodologie, tableaux

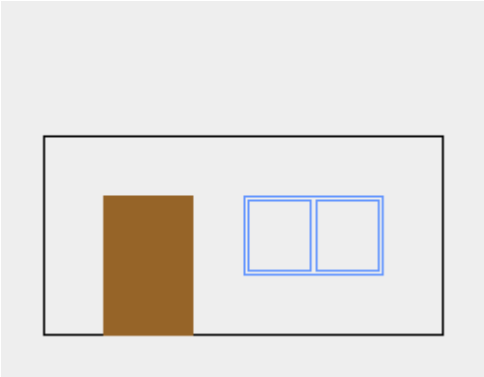
- Résumé des épisodes précédents
- Correction du QCM1 / présentation des DM1 & 2
- Compétences acquises en INF120
- Méthodologie pour l'algorithmique
- Exemples d'application de la méthodologie
- Les tableaux

Le toit avec une pente à 26,565°



```
i=0;
rouge=couleurRGB(200,0,0);
while (i<51){
   dessineLigne(g,300+2*i,199-i,300+2*i,199,rouge);
   dessineLigne(g,300+2*i+1,199-i,300+2*i+1,199,rouge);
   dessineLigne(g,500-2*i,199-i,500-2*i,199,rouge);
   dessineLigne(g,500-2*i+1,199-i,500-2*i+1,199,rouge);
   i=i+1;
}
```

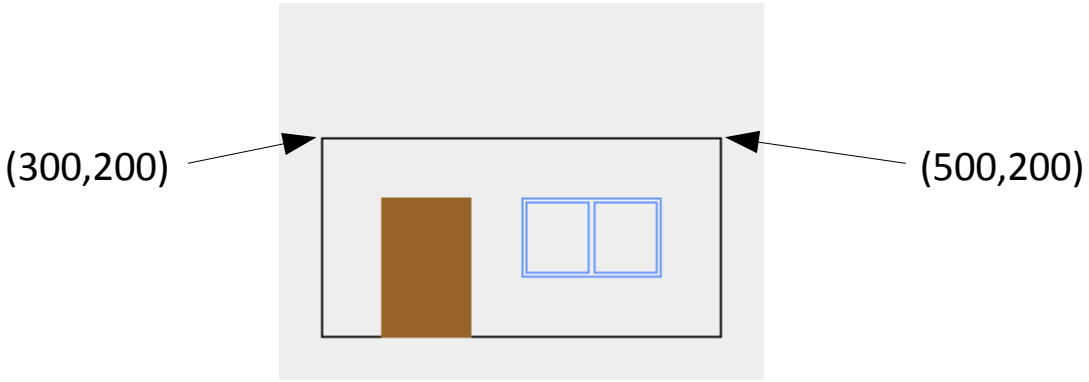
Le toit avec une pente à 26,565°



```
i=0;
rouge=couleurRGB(200,0,0);
while (i<51){
   dessineLigne(g,300+2*i,199-i,300+2*i,199,rouge);
   dessineLigne(g,300+2*i+1,199-i,300+2*i+1,199,rouge);
   dessineLigne(g,500-2*i,199-i,500-2*i,199,rouge);
   dessineLigne(g,500-2*i+1,199-i,500-2*i+1,199,rouge);
   i=i+1;
}
```

Faire la trace de l'algorithme

Le toit avec une pente à 26,565°

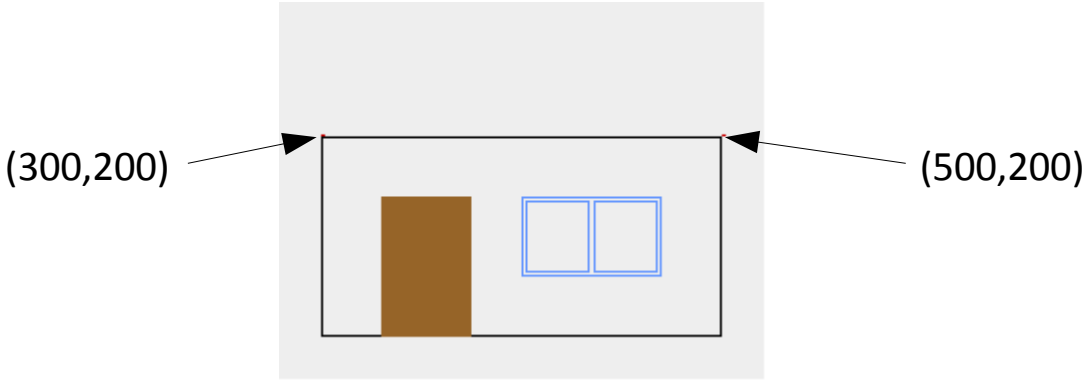



```
i=0;
rouge=couleurRGB(200,0,0);
while (i<51){
   dessineLigne(g,300+2*i,199-i,300+2*i,199,rouge);
   dessineLigne(g,300+2*i+1,199-i,300+2*i+1,199,rouge);
   dessineLigne(g,500-2*i,199-i,500-2*i,199,rouge);
   dessineLigne(g,500-2*i+1,199-i,500-2*i+1,199,rouge);
   i=i+1;
}
```

Faire la trace de l'algorithme

“dessineLigne(Graphics g, int abscisse1, int ordonnee1, int abscisse2, int ordonnee2, Color couleur), qui dessine sur l'objet g une ligne commençant au point de coordonnées (abscisse1,ordonnee1) et se terminant au point de coordonnées (abscisse2,ordonnee2).”

Le toit avec une pente à 26,565°




```
i=0;
rouge=couleurRGB(200,0,0);
while (i<51){
   dessineLigne(g,300+2*i,199-i,300+2*i,199,rouge);
   dessineLigne(g,300+2*i+1,199-i,300+2*i+1,199,rouge);
   dessineLigne(g,500-2*i,199-i,500-2*i,199,rouge);
   dessineLigne(g,500-2*i+1,199-i,500-2*i+1,199,rouge);
   i=i+1;
}
```

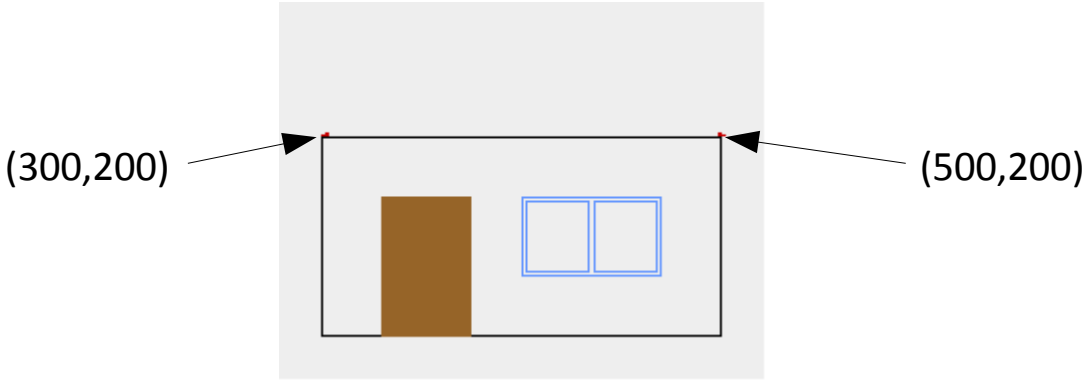
Faire la trace de l'algorithme : *i*=0 :

dessineLigne(g,300,199,300,199,rouge)

dessineLigne(g,301,199,301,199,rouge)

...

Le toit avec une pente à 26,565°




```
i=0;
rouge=couleurRGB(200,0,0);
while (i<51){
   dessineLigne(g,300+2*i,199-i,300+2*i,199,rouge);
   dessineLigne(g,300+2*i+1,199-i,300+2*i+1,199,rouge);
   dessineLigne(g,500-2*i,199-i,500-2*i,199,rouge);
   dessineLigne(g,500-2*i+1,199-i,500-2*i+1,199,rouge);
   i=i+1;
}
```

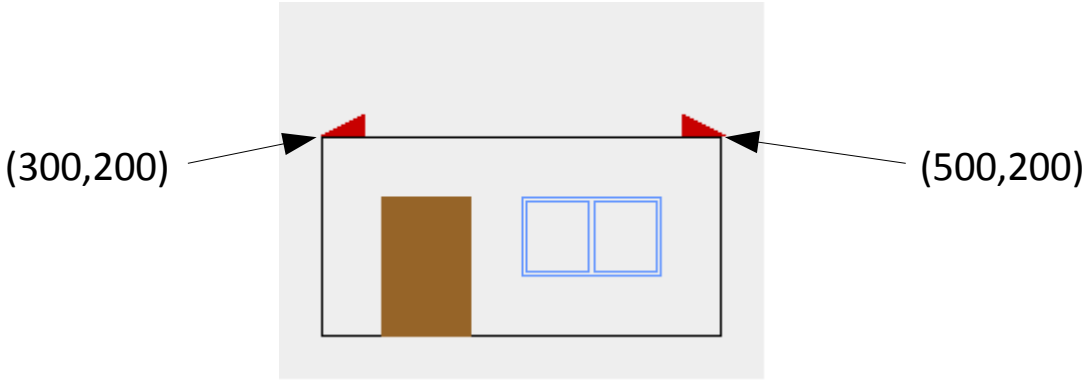
Faire la trace de l'algorithme : *i*=1

dessineLigne(g,302,198,302,199,rouge)

dessineLigne(g,303,198,303,199,rouge)

...

Le toit avec une pente à 26,565°




```
i=0;
rouge=couleurRGB(200,0,0);
while (i<51){
   dessineLigne(g,300+2*i,199-i,300+2*i,199,rouge);
   dessineLigne(g,300+2*i+1,199-i,300+2*i+1,199,rouge);
   dessineLigne(g,500-2*i,199-i,500-2*i,199,rouge);
   dessineLigne(g,500-2*i+1,199-i,500-2*i+1,199,rouge);
   i=i+1;
}
```

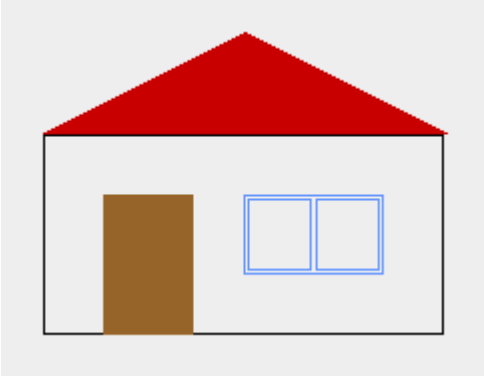
Faire la trace de l'algorithme : *i*=10

dessineLigne(g,320,189,320,199,rouge)

dessineLigne(g,321,189,321,199,rouge)

...

Le toit avec une pente à 26,565°



```
i=0;
rouge=couleurRGB(200,0,0);
while (i<51){
   dessineLigne(g,300+2*i,199-i,300+2*i,199,rouge);
   dessineLigne(g,300+2*i+1,199-i,300+2*i+1,199,rouge);
   dessineLigne(g,500-2*i,199-i,500-2*i,199,rouge);
   dessineLigne(g,500-2*i+1,199-i,500-2*i+1,199,rouge);
   i=i+1;
}
```

On sort de la boucle après avoir affecté la valeur 51 à la variable *i*

Le toit avec une pente à 26,565°

```
i=0;
rouge=couleurRGB(200,0,0);
while (i<51){
   dessineLigne(g,300+2*i,199-i,300+2*i,199,rouge);
   dessineLigne(g,300+2*i+1,199-i,300+2*i+1,199,rouge);
   dessineLigne(g,500-2*i,199-i,500-2*i,199,rouge);
   dessineLigne(g,500-2*i+1,199-i,500-2*i+1,199,rouge);
   i=i+1;
}
```

Traduction en pseudo-code :

Algorithme **DessineToit**()
Variables : entier *i*, couleur *rouge*
Début
    i ← 0
    rouge ← **couleurRGB**(200,0,0)
    Tant que *i*<51 faire :
        dessineLigne(300+2*i*,199-*i*,300+2*i*,199,rouge)
        dessineLigne(301+2*i*,199-*i*,301+2*i*,199,rouge)
        dessineLigne(500-2*i*,199-*i*,500-2*i*,199,rouge)
        dessineLigne(501-2*i*,199-*i*,501-2*i*,199,rouge)
        i ← *i*+1
    Fin TantQue
Fin

Exercice 1 du TD2

• 42 en binaire : 101010
• 84 en binaire : 1010100
• Déduisez-en un algorithme **DiviseParDeuxPair** qui prend en entrée une chaîne de caractères qui contient un nombre binaire *n* et s'il est pair, renvoie une chaîne de caractères qui contient la valeur de *n*/2 écrite en binaire. Vous utiliserez l'algorithme **Caractère** qui prend en entrée une chaîne de caractères *chaine* et un entier *i* et renvoie le *i*-ième caractère de chaine, ainsi que l'algorithme **SousChaine** qui renvoie la partie de la chaîne de caractères *chaine* allant du *i*-ième au *j*-ième caractère (inclus), et l'algorithme **Longueur** qui renvoie le nombre de caractères de la chaîne de caractères *chaine*.

Le nombre d'utilisations d'un mot dans un texte

• Écrivez un algorithme **TrouveMot** qui prend en entrée une chaîne de caractères *mot* et une chaîne de caractères *texte*, et compte le nombre de fois que *mot* apparaît dans *texte*.

Le nombre d'utilisations d'un mot dans un texte

• Écrivez un algorithme **TrouveMot** qui prend en entrée une chaîne de caractères *mot* et une chaîne de caractères *texte*, et compte le nombre de fois que *mot* apparaît dans *texte*.

Exemple : *mot* = “fuck”

texte=“ Nobody's fuckin' screaming, Craig! Wake the fuck up! ”

South Park S15E09
http://www.spscriptorium.com/Season15/E1509script.htm

Le nombre d'utilisations d'un mot dans un texte

• Écrivez un algorithme **TrouveMot** qui prend en entrée une chaîne de caractères *mot* et une chaîne de caractères *texte*, et compte le nombre de fois que *mot* apparaît dans *texte*.

Exemple : *mot* = “fuck”

texte=“ Nobody's fuckin' screaming, Craig! Wake the fuck up! ”

Le nombre d'utilisations d'un mot dans un texte

• Écrivez un algorithme **TrouveMot** qui prend en entrée une chaîne de caractères *mot* et une chaîne de caractères *texte*, et compte le nombre de fois que *mot* apparaît dans *texte*.

Exemple : *mot* = “fuck”

texte=“ Nobody's fuckin' screaming, Craig! Wake the fuck up! ”

Idée : regarder à toutes les positions de la chaîne de caractères *texte* si on trouve le mot *mot*.

Décomposition du problème :

• à toutes les positions de *texte*

• on teste si on trouve *mot*

Le nombre d'utilisations d'un mot dans un texte

• Écrivez un algorithme **TrouveMot** qui prend en entrée une chaîne de caractères *mot* et une chaîne de caractères *texte*, et compte le nombre de fois que *mot* apparaît dans *texte*.

Exemple : *mot* = “fuck”

texte=“ Nobody's fuckin' screaming, Craig! Wake the fuck up! ”

Idée : regarder à toutes les positions de la chaîne de caractères *texte* si on trouve le mot *mot*.

Décomposition du problème :

• à toutes les positions de *texte*

• on teste si on trouve *mot*

la sous-chaine de *texte* qui commence au *i*-ième caractère et a autant de lettres que *mot*, c'est-à-dire 4, est-elle égale à *mot* ?

reformulation du problème avec **Caractères**, **SousChaine**, **Longueur**

Le nombre d'utilisations d'un mot dans un texte

• Écrivez un algorithme **TrouveMot** qui prend en entrée une chaîne de caractères *mot* et une chaîne de caractères *texte*, et compte le nombre de fois que *mot* apparaît dans *texte*.

Exemple : *mot* = “fuck”

texte=“ Nobody's fuckin' screaming, Craig! Wake the fuck up! ”

Idée : regarder à toutes les positions de la chaîne de caractères *texte* si on trouve le mot *mot*.

Décomposition du problème :

• à toutes les positions de *texte*

• on teste si on trouve *mot*

la sous-chaine de *texte* qui commence au *i*-ième caractère et a autant de lettres que *mot*, c'est-à-dire 4, est-elle égale à *mot* ?

mot = **SousChaine**(*texte*,*i*,*i*+3) ?

reformulation du problème avec **Caractères**, **SousChaine**, **Longueur**

Le nombre d'utilisations d'un mot dans un texte

• Écrivez un algorithme **TrouveMot** qui prend en entrée une chaîne de caractères *mot* et une chaîne de caractères *texte*, et compte le nombre de fois que *mot* apparaît dans *texte*.

Exemple : *mot* = “fuck”

texte=“ Nobody's fuckin' screaming, Craig! Wake the fuck up! ”

Idée : regarder à toutes les positions de la chaîne de caractères *texte* si on trouve le mot *mot*.

Décomposition du problème :

• à toutes les positions de *texte*

• on teste si on trouve *mot*

la sous-chaine de *texte* qui commence au *i*-ième caractère et a autant de lettres que *mot*, c'est-à-dire 4, est-elle égale à *mot* ?

mot = **SousChaine**(*texte*,*i*,*i*+**Longueur**(*mot*)-1) ?

reformulation du problème avec **Caractères**, **SousChaine**, **Longueur**

Le nombre d'utilisations d'un mot dans un texte

• Écrivez un algorithme **TrouveMot** qui prend en entrée une chaîne de caractères *mot* et une chaîne de caractères *texte*, et compte le nombre de fois que *mot* apparaît dans *texte*.

Exemple : *mot* = “fuck”

texte=“ Nobody's fuckin' screaming, Craig! Wake the fuck up! ”

Idée : regarder à toutes les positions de la chaîne de caractères *texte* si on trouve le mot *mot*.

Décomposition du problème :

• à toutes les positions de *texte*

• on teste si on trouve *mot*

appeler cette position *i*
la faire varier de 1 à ...

mot = **SousChaine**(*texte*,*i*,*i*+**Longueur**(*mot*)-1) ?

reformulation du problème avec **Caractères**, **SousChaine**, **Longueur**

Le nombre d'utilisations d'un mot dans un texte

• Écrivez un algorithme **TrouveMot** qui prend en entrée une chaîne de caractères *mot* et une chaîne de caractères *texte*, et compte le nombre de fois que *mot* apparaît dans *texte*.

Exemple : *mot* = “fuck”

texte=“ Nobody's fuckin' screaming, Craig! Wake the fuck up! ”

Idée : regarder à toutes les positions de la chaîne de caractères *texte* si on trouve le mot *mot*.

Décomposition du problème :

• à toutes les positions de *texte*

• on teste si on trouve *mot*

appeler cette position *i*
la faire varier de 1 à ...

i ← 1
Tant que *i*<**Longueur**(*texte*)-**Longueur**(*mot*)+2 faire : ...

mot = **SousChaine**(*texte*,*i*,*i*+**Longueur**(*mot*)-1) ?

reformulation du problème avec **Caractères**, **SousChaine**, **Longueur**

Le nombre d'utilisations d'un mot dans un texte

• Écrivez un algorithme **TrouveMot** qui prend en entrée une chaîne de caractères *mot* et une chaîne de caractères *texte*, et compte le nombre de fois que *mot* apparaît dans *texte*.

Exemple : *mot* = “fuck”

texte=“ Nobody's fuckin' screaming, Craig! Wake the fuck up! ”

Idée : regarder à toutes les positions de la chaîne de caractères *texte* si on trouve le mot *mot*.

Décomposition du problème :

• à toutes les positions de *texte*

• on teste si on trouve *mot*

i ← 1
Tant que *i*<**Longueur**(*texte*)-**Longueur**(*mot*)+2 faire : ...

mot = **SousChaine**(*texte*,*i*,*i*+**Longueur**(*mot*)-1) ?

reformulation du problème avec **Caractères**, **SousChaine**, **Longueur**

Le nombre d'utilisations d'un mot dans un texte

• Écrivez un algorithme **TrouveMot** qui prend en entrée une chaîne de caractères *mot* et une chaîne de caractères *texte*, et compte le nombre de fois que *mot* apparaît dans *texte*.

Exemple : *mot* = “fuck”

texte=“ Nobody's fuckin' screaming, Craig! Wake the fuck up! ”

i ← 1
Tant que *i*<**Longueur**(*texte*)-**Longueur**(*mot*)+2 faire :

mot = **SousChaine**(*texte*,*i*,*i*+**Longueur**(*mot*)-1) ?

Le nombre d'utilisations d'un mot dans un texte

• Écrivez un algorithme **TrouveMot** qui prend en entrée une chaîne de caractères *mot* et une chaîne de caractères *texte*, et compte le nombre de fois que *mot* apparaît dans *texte*.

Exemple : *mot* = “fuck”

texte=“ Nobody's fuckin' screaming, Craig! Wake the fuck up! ”

Algorithme **TrouveMot** :
Entrées : chaîne de caractères *mot*, chaîne de caractères *texte*
Sorties : nombre de fois (entier) que *mot* apparaît dans *texte*
Début
    i ← 1
    compteur ← 0
    Tant que *i*<**Longueur**(*texte*)-**Longueur**(*mot*)+2 faire :
        Si *mot* = **SousChaine**(*texte*,*i*,*i*+**Longueur**(*mot*)-1) alors :
            compteur ← *compteur*+1
        i ← *i*+1
        FinSi
    Fin TantQue
    renvoyer *compteur*
Fin

Le nombre d'utilisations d'un mot dans un texte

La “minute culturelle”

Comment compter **plusieurs mots** dans un texte ?

Méthode naïve :
appliquer l'algorithme **TrouveMot** pour chaque mot à compter
➡ autant de lectures du texte que de mots distincts à compter

Le nombre d'utilisations d'un mot dans un texte

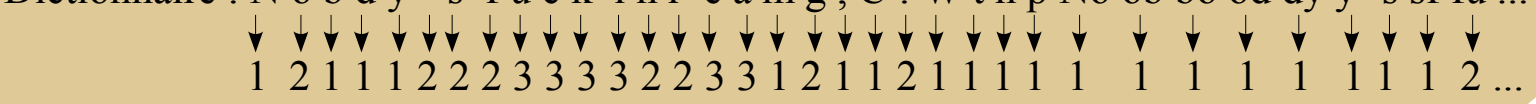
La "minute culturelle"

Comment compter **plusieurs mots** dans un texte ?

Méthode naïve :
appliquer l'algorithme **TrouveMot** pour chaque mot à compter
➡ autant de lectures du texte que de mots distincts à compter

Méthode astucieuse **plus rapide** :
faire une lecture du texte en construisant un dictionnaire
➡ compter tous les mots au fur et à mesure
➡ ensuite, pour avoir le nombre d'apparitions d'un mot,
simplement aller voir dans le dictionnaire
(pas besoin de reparcourir le texte)

Texte : "Nobody's fuckin' screaming, Craig! Wake the fuck up!"



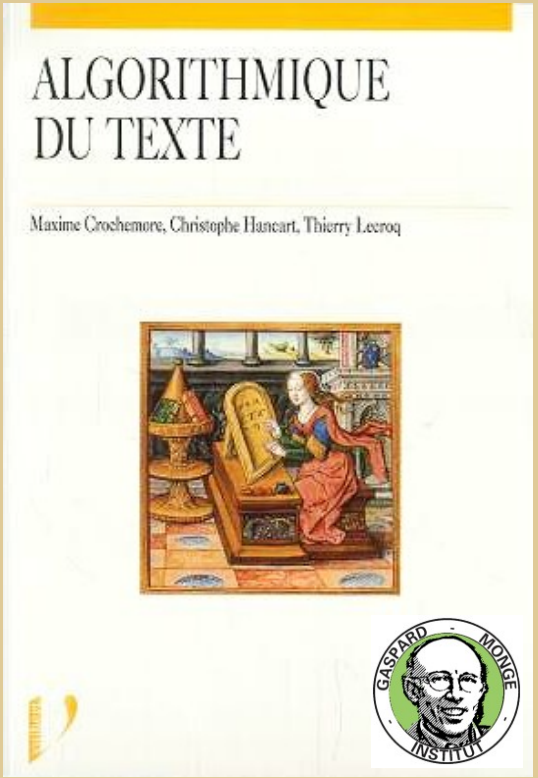
Le nombre d'utilisations d'un mot dans un texte

La “minute culturelle”

Comment compter **plusieurs mots** dans un texte ?

Problèmes d'“Algorithmique du texte”

➡ des experts au LIGM dans le bâtiment Copernic !

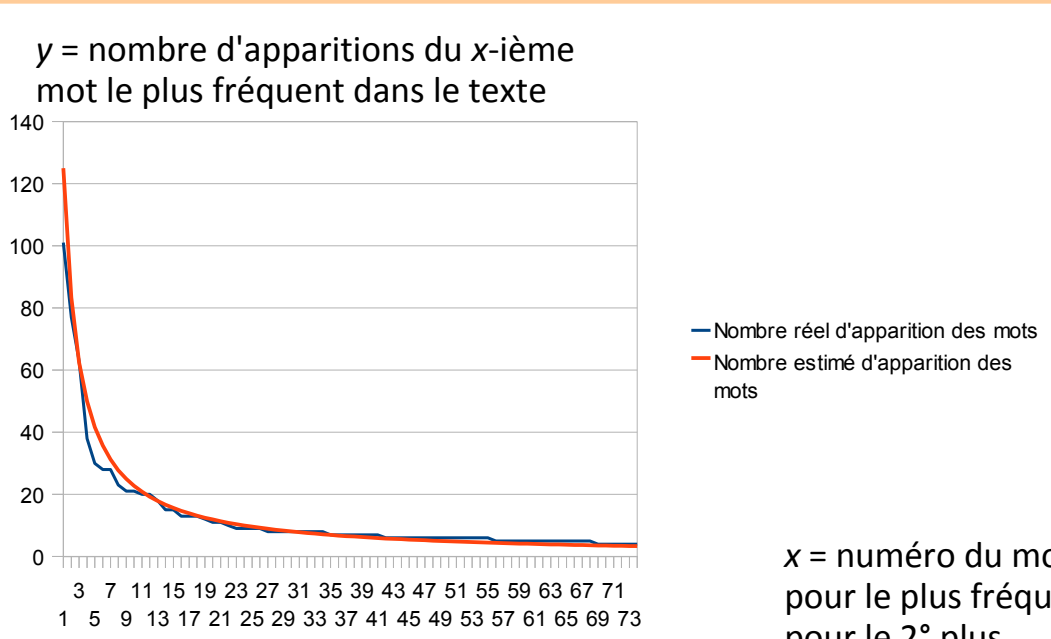


http://igm.univ-mlv.fr/~mac/CHL/CHL.html

Le nombre d'utilisations d'un mot dans un texte

La “minute mathématique”

La loi de Zipf prédit la courbe du nombre d'utilisations des mots les plus fréquents d'un texte.



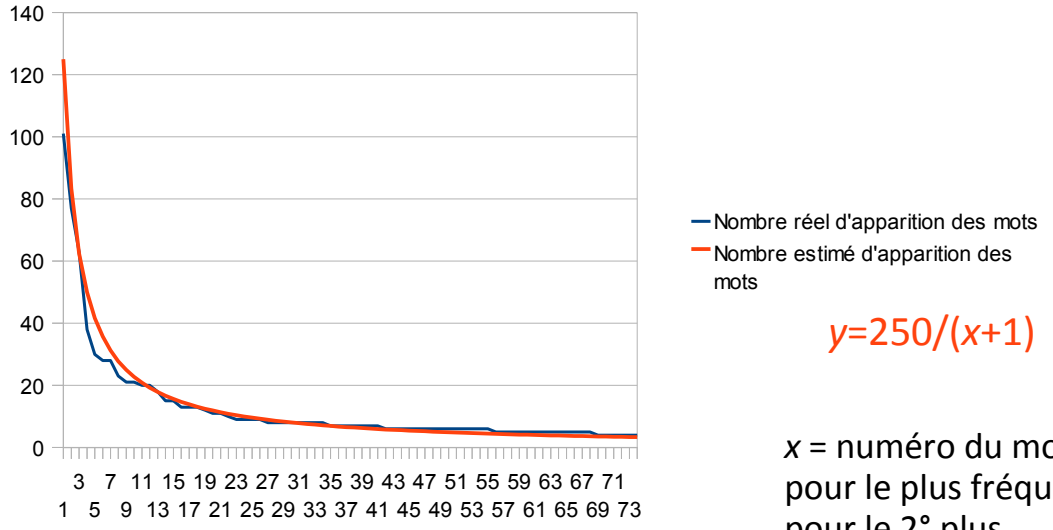
x = numéro du mot (1 pour le plus fréquent, 2 pour le 2° plus fréquent...)

Le nombre d'utilisations d'un mot dans un texte

La “minute mathématique”

La loi de Zipf prédit la courbe du nombre d'utilisations des mots les plus fréquents d'un texte.

y = nombre d'apparitions du *x*-ième mot le plus fréquent dans le texte



$y=250/(x+1)$

x = numéro du mot (1 pour le plus fréquent, 2 pour le 2° plus fréquent...)

Fonctionne pour n'importe quel texte assez long...

Plan du cours 3 – Méthodologie, tableaux

- Résumé des épisodes précédents
- Correction du QCM1 / présentation des DM1 & 2
- Compétences acquises en INF120
- Méthodologie pour l'algorithmique
- Exemples d'application de la méthodologie
- Les tableaux

Les tableaux

Les tableaux sont des variables qui contiennent **plusieurs variables de même type**, stockées chacune dans une des cases du tableau.

Par exemple,

Un **tableau d'entiers** :

4
5
1
23
8
9

Un **tableau de chaînes de caractères** :

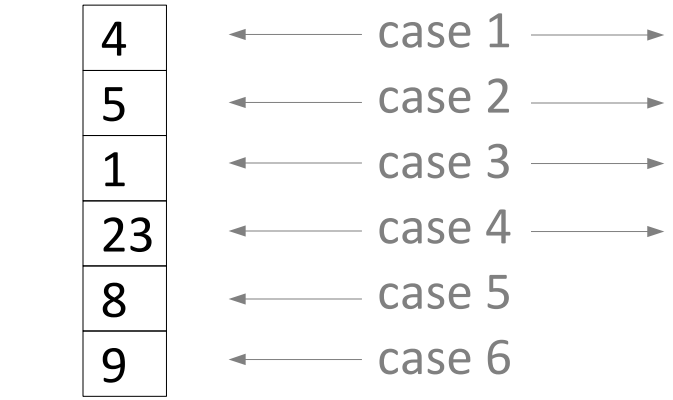
“chaine1”
“chaine2”
“blabla”
“toto”

Les tableaux

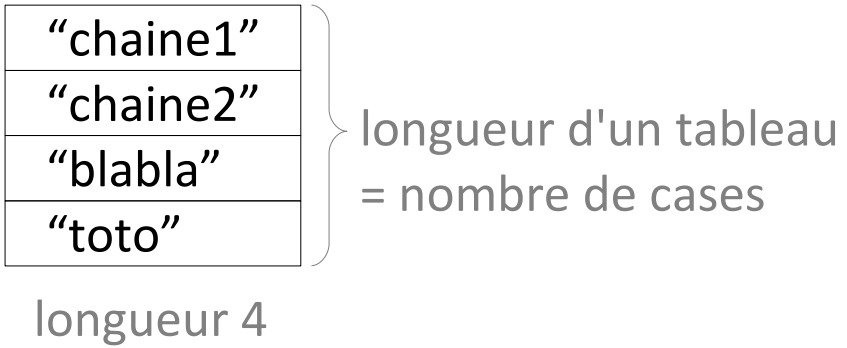
Les tableaux sont des variables qui contiennent **plusieurs variables de même type**, stockées chacune dans une des cases du tableau.

Par exemple,

Un **tableau d'entiers** :



Un **tableau de chaînes de caractères** :



Les tableaux

en pseudo-code

Les tableaux sont des variables qui contiennent **plusieurs variables de même type**, stockées chacune dans une des cases du tableau.

Par exemple, Variables : *tableau1*, un tableau d'entiers,
tableau2, un tableau de chaînes de caractères

Un **tableau d'entiers** :

4	case 1
5	case 2
1	case 3
23	case 4
8	case 5
9	case 6

longueur 6

tableau1=**NouveauTableau**(6)
Case(*tableau1*,1)←4
Case(*tableau1*,2)←5
Case(*tableau1*,3)←1
Case(*tableau1*,4)←23
Case(*tableau1*,5)←8
Case(*tableau1*,6)←9

Un **tableau de chaînes de caractères** :

“chaine1”
“chaine2”
“blabla”
“toto”

longueur 4

longueur d'un tableau
= nombre de cases
Longueur(*tableau2*)

tableau2=**NouveauTableau**(4)
Case(*tableau2*,1)←”chaine1”
Case(*tableau2*,2)←”chaine2”
Case(*tableau2*,3)←”blabla”
Case(*tableau2*,4)←”toto”

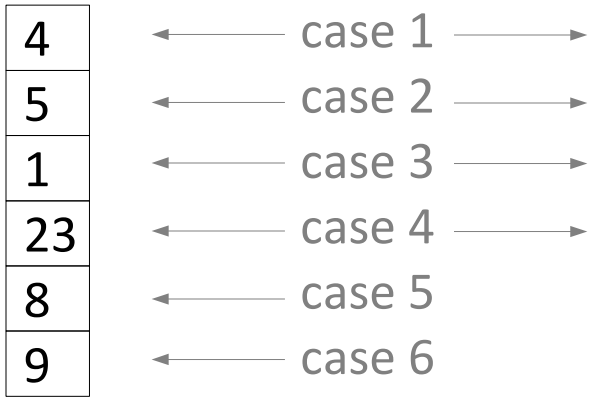
Les tableaux

en Java

Les tableaux sont des variables qui contiennent **plusieurs variables de même type**, stockées chacune dans une des cases du tableau.

Par exemple, int[] *tableau1*; String[] *tableau2*;

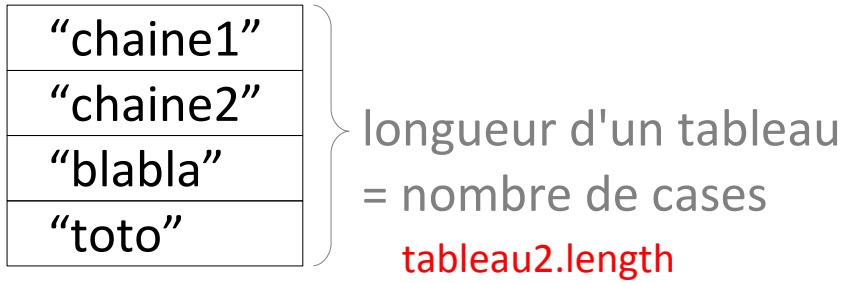
Un **tableau d'entiers** :



longueur 6

```
tableau1=new int[6];
tableau1[0]=4;
tableau1[1]=5;
tableau1[2]=1;
tableau1[3]=23;
tableau1[4]=8;
tableau1[5]=9;
```

Un **tableau de chaînes de caractères** :



longueur 4

```
tableau2=new String[4];
tableau2[0]=”chaine1”;
tableau2[1]=”chaine2”;
tableau2[2]=”blabla”;
tableau2[3]=”toto”;
```

Attention, cases du tableau `t` numérotées de `0` à `t.length-1` en Java.

Les tableaux

Pour lire le contenu d'un tableau...
il faut une boucle pour aller lire chaque case !

Si le tableau a été prévu trop court au début, impossible de changer sa longueur... il faut une boucle pour le recopier dans un tableau plus grand !

Manipulation et expériences en TD/TP...