Compétences acquises en INF120

Connaître les éléments de base d'un algorithme *en pseudo-code*

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

Connaître les éléments de base d'un algorithme *en Java*

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.

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

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...)

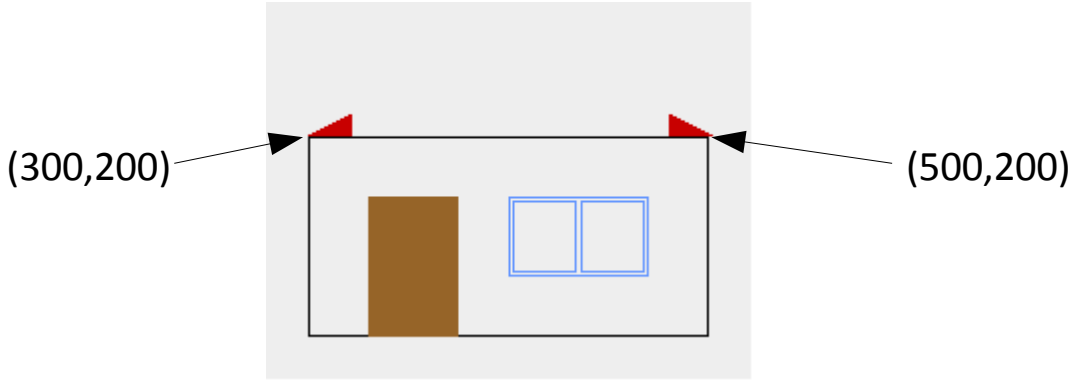
Ensuite, résoudre le problème en utilisant ces outils :
- comment résoudre le problème **étape par étape** ? (essayer sur l'exemple testé)
- est-ce que les **outils à disposition** sont **utilisables** pour réaliser chaque étape ?

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)

Le toit avec une pente à 26,565°

Instructions Java :

```
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

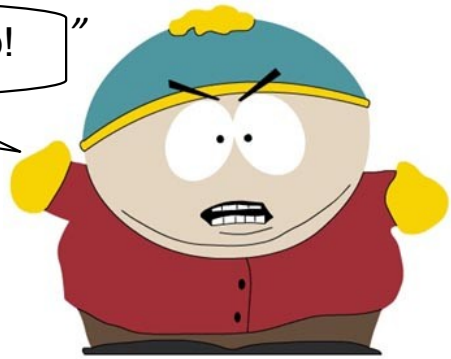
Faire la trace de l'algorithme : *i*=10
dessineLigne(g,320,189,320,199,rouge)
dessineLigne(g,321,189,321,199,rouge)
...

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* = “f***”

texte=“ Nobody's f***in' screaming, Craig! Wake the f*** 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
        FinSi
        i ← *i*+1
    Fin TantQue
    renvoyer *compteur*
Fin

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)

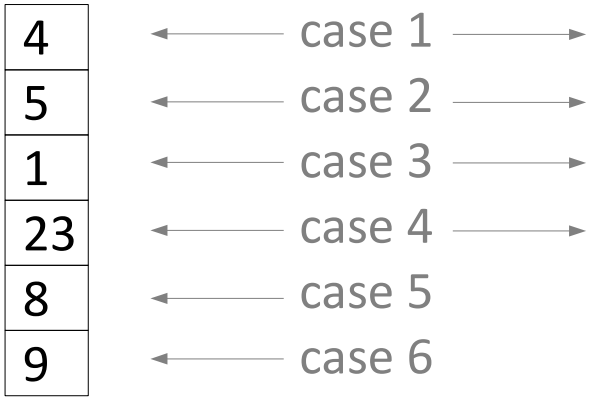
Les tableaux

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

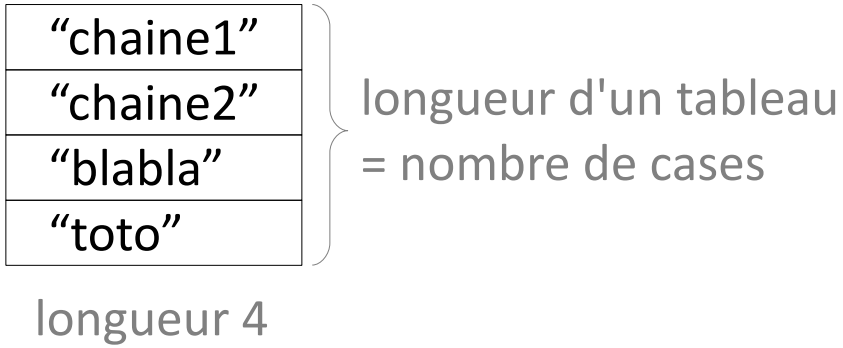
en pseudo-code

Par exemple,

Un **tableau d'entiers** :



Un **tableau de chaînes de caractères** :



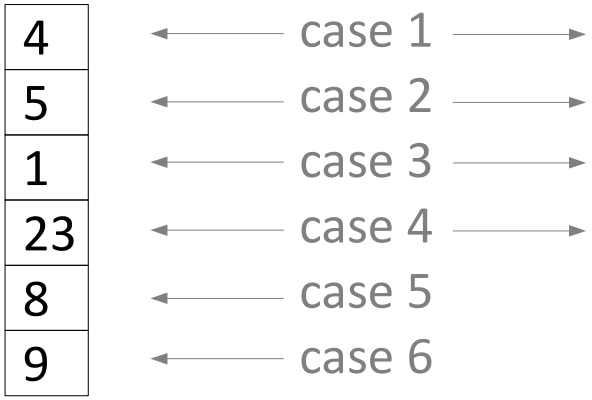
Les tableaux

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

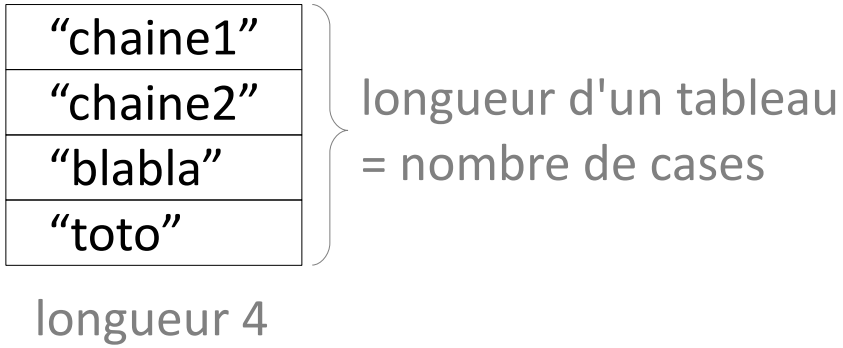
en Java

Par exemple,

Un **tableau d'entiers** :



Un **tableau de chaînes de caractères** :



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