Principales instructions

Gilles Trombettoni

IUT MPL-Sète, département info

Développement initiatique

Septembre 2021

Petit jeu d'instructions de base dans les langages de programmation

Les algorithmes (et les langages de programmation) offrent un jeu très restreint d'instructions de base. L'algorithmique est l'art de savoir manier ce petit vocabulaire pour exprimer toutes les méthodes imaginables. « Vocabulaire » :

- entrées-sorties ;
- variables et **structures de données** plus sophistiquées (dont les tableaux) ;
- affectations de variables ;
- instructions conditionnelles (`si alors sinon`) ;
- répétition d'un ensemble d'instructions (boucles) ;
- appel à des sous-programmes (procédures, fonctions, méthodes ; une infinité de sous-programmes existent).

Communication avec l'être humain

Sortie à l'écran

- Syntaxe : `afficher (...)`
- Exemple :

```
afficher("Double du prix = ", 2*prix, " euros")
```

Entrée au clavier

- Syntaxe : `saisir(<variable>)`
  1. Le programme se bloque et attend une valeur saisie au clavier, suivie de entrée.
  2. La valeur saisie est affectée à la variable en argument.
- Par défaut, on suppose que l'utilisateur saisit une valeur du bon type.
- Exemple :

```
afficher("Donner un prix")
saisir(prix)
afficher ("Double du prix = ", 2*prix, " euros")
```

Pourquoi les termes *entrées* et *sorties* ?

Indice : soyez ordinato-centré !

Qu'est-ce qu'une variable ?

- Définition : Une **variable** est un symbole, un nom associé à une information.

  A une variable correspond un emplacement mémoire (RAM) permettant de stocker le résultat de calculs (intermédiaires) d'un algorithme/programme.

- Une variable a généralement un **type**, défini **statiquement** (une fois pour toutes, dans le fichier source) ou **dynamiquement** (pendant l'exécution de l'algorithme : une variable typée dynamiquement peut donc changer de type au cours de l'exécution du programme...), selon le langage.

- Types principaux : entier (relatifs), réel (flottant, double), booléen, caractère, chaînes de caractères.

Taille des variables en mémoire

La taille de l'emplacement mémoire d'une variable dépend de son type, du langage, de l'ordinateur.

Par exemple :

- entier : 4 octets
- double : 8 ou 10 octets
- booléen : 1 bit !
- caractère : 7 bits (norme ASCII), 1 octet (norme ISO 8859), ou 2 ou 3 octets (normes Unicode UTF-16 et UTF-8)
- chaîne de caractères ?

Déclaration des variables en langage algorithmique maison

Nous considérons des variables définies **statiquement**. Exemple :

```
Algo demoVariables
Variables
    n : entier
    x,y : reel
    estFini : booleen
    lettre : caractere
    mot : chaine (de caracteres)
Debut
    ...
Fin demoVariables
```

Ecriture d'une variable (affectation)

`<variable>` ← `<valeur>` affecte `<valeur>` à `<variable>`.

Remarque : Si la variable contenait déjà une valeur, cette dernière est modifiée (écrasée).

Exemple

```
Algo demoVariables
Variables
  ...
Debut
  n  <-  666
  lettre <- 'z'
  mot <- "omega"
  estFini <- vrai
  ...
Fin demoVariables
```

Lecture d'une variable

Exemple

```
Algo demoVariables
Variables
  ...
Debut
  ...
  si (n div 2 == 333) et (lettre == 'z') et estFini
  alors
    afficher ("This is the end, Beautiful friend")
  finSi
Fin demoVariables
```

Remarque : dans certains langages, l'accès à la valeur de la variable se fait à l'aide d'une syntaxe particulière (ex : `${mot}`).

Tableaux

Les tableaux sont des variables **composites**, qui donnent accès à plusieurs valeurs. Ce sont les **structures de données** les plus utilisées en informatique.

Définition (wikipedia)

Un tableau est une séquence finie d'éléments auxquels on peut accéder efficacement par leur position, ou **indice**, dans la séquence. Un tableau « de base » contient des valeurs qui sont toutes du même type.

3.14	2.72	1.62	-1.655	-10.54
0	1	2	3	4

Selon le langage, les tableaux sont indicés par des entiers commençant à 0 (ex : C, Java), des entiers commençant à 1 (ex : Fortran, Matlab) ou par n'importe quel type énuméré (ex : Ada).

Les tableaux dans le langage maison

Déclaration (dans la partie `Variables`)

- `<nom de tableau> : tableau de <taille> <type des éléments>` définit un tableau de `<taille>` éléments entiers indicés de 0 à `<taille>`−1.
- Exemple d'un tableau de 10 caractères :

```
tab : tableau de 10 caractères
```

Modification d'une case du tableau

- `<nom de tableau>[<indice>]` ← `<valeur>`
- ex : `t[2]` ← `'z'`

En fait, un élément (ou case) d'un tableau (ex : `t[5]`) se manipule en lecture et en écriture comme une variable d'un type de base (ex : `x`).

Les tableaux en mémoire

- Un tableau est stocké en mémoire vive dans une zone contiguë.
- Un tableau est du coup déterminé par l'adresse de sa première case.
- L'accès à chaque case prend le même temps de calcul (premier élément ou 1000^e élément), très rapide : une multiplication, une addition, un accès à une adresse mémoire !

Instruction « si alors sinon »

Syntaxe

```
si <condition> alors
   <bloc d'instructions 1>
sinon
   <bloc d'instructions 2>
finSi
```

Sémantique

L'un des deux blocs d'instructions est exécuté selon que la condition est vraie ou fausse.

Exemple

```
si x < 100 et lettre == 'a' alors
   afficher("Exemple idiot")
   n <- 8
sinon
   afficher("Exemple tres idiot")
   n <- 5
finSi
```

Conditions

- Condition ≡ expression booléenne
- Opérateurs : $==$, $<$, $\leq$, $>$, $\geq$, $\neq$, `et`, `ou`, `non`.
- Exemple : `x < 100 et lettre == 'a' ou non estFini`
  L'opérateur `et` est prioritaire sur l'opérateur `ou`.

Cf. cours de logique

Il existe dans la plupart des langages de programmation des opérateurs `Et` et `Ou` qui sont **séquentiels**.

Opérateur `<condition 1> EtAlors <condition 2>`

```
si <condition 1> == faux alors faux
sinon // <condition 1> == vrai
   <condition 2>
finSi
```

Opérateur `<condition 1> OuSinon <condition 2>`

```
si <condition 1> == vrai alors vrai
sinon // <condition 1> == faux
   <condition 2>
finSi
```

Instruction « si alors »

L'instruction conditionnelle peut être incomplète.

Syntaxe

```
si <condition> alors
   <bloc d'instructions 1>
finSi
```

Sémantique

Le bloc d'instructions est exécuté si et seulement si la condition est vraie. Sinon, rien n'est exécuté et on passe à l'instruction suivante.

Exemple

```
si x < 100 et lettre == 'a' alors
   afficher("Exemple idiot")
   n <- 8
finSi
afficher("Suite...")
```

Instruction « si alors sinonSi sinon »

Certains langages autorisent cette instruction qui équivaut à plusieurs instructions « si alors sinon ».

Syntaxe

```
si <condition 1> alors
   <bloc d'instructions 1>
sinonSi <condition 2> alors
   <bloc d'instructions 2>
sinonSi <condition 3> alors
   <bloc d'instructions 3>
...
sinon // Ttes les conditions au dessus sont fausses
   <bloc d'instructions n+1>
finSi
```

Instruction « si alors sinonSi sinon »

Instruction équivalente

```
si <condition 1> alors
   <bloc d'instructions 1>
sinon // Condition 1 est fausse
   si <condition 2> alors
      <bloc d'instructions 2>
   sinon // Conditions 1 et 2 sont fausses
      si <condition 3> alors
         <bloc d'instructions 3>
      ...
      sinon // Ttes les conditions sont fausses
         <bloc d'instructions n+1>
      finSi
   finSi
finSi
```

Instructions répétitives (boucles)

Plusieurs instructions répétitives permettent de répéter un **bloc** (une suite) d'instructions plusieurs fois :

- boucle `tantQue`
  (la plus utilisée et qui permet de presque tout faire)
- boucle `faire tantQue` (ou `répéter jusqu'à`)
- boucle `pour`
- boucle « générale » (appelons-la `boucle`)

Boucle `tantQue`

Syntaxe

```
tantQue <condition> faire
   <bloc d'instructions>
finTantQue
```

Sémantique

Le bloc d'instructions est exécuté tant que la condition (de continuation de la boucle) est vraie.

Exemple (chercher un élément donné dans un tableau de 100 cases)

```
i <- 0 ; trouve <- faux
tantQue i < 100 et non trouve faire
   si t[i] == elt alors // elt trouve
      trouve <- vrai
   sinon
      i <- i+1
   finSi
finTantQue
afficher ("Element", elt, "trouve ? Rep = ", trouve)
```

Boucle `faire tantQue`

Syntaxe

```
faire
|   <bloc d'instructions>
tantQue <condition>
```

Sémantique

Le bloc d'instructions est exécuté **au moins une fois**
tant que la condition (de continuation de la boucle) est vraie.

Exemple (redemander une saisie tant que non correcte)

```
faire
|  afficher("Entrer un nombre strictement positif")
|  saisir (nb)
tantQue nb <= 0
afficher ("Nombre saisi = ", nb)
```

Remarque : des langages proposent la boucle similaire :

`repeter <instructions> jusque <condition d'arret>`

Boucle `pour`

La boucle `pour` permet d'itérer sur des ensembles (totalement) ordonnés, des « collections séquentielles ».

Syntaxe

```
pour elt dans collectionSequentielle faire
    <instructions de traitement de elt>
finPour
```

Sémantique

A chaque itération, la variable `elt` prend comme valeur un élément de la collection, dans l'ordre de cette séquence.

Exemple de boucle pour en Java

```
String [] tabMots = {"Je", "suis", "content"};
for (String s : tabMots) {
    System.out.println(s);
}
```

Boucle `pour` sur des intervalles d'entiers

On peut a fortiori (et historiquement) utiliser une boucle `pour` sur des intervalles d'entiers.

`pour` simple (exemple)

```
pour i dans 20..29 faire
    afficher(" ", i)
finPour
→ 20 21 22 23 24 25 26 27 28 29
```

`pour` avec incrémention d'un pas supérieur à 1 (exemple)

```
pour i dans 20..29 par pas de 2 faire
    afficher(" ", i)
finPour
→ 20 22 24 26 28
```

`pour` en sens inverse (exemple)

```
pour i dans 20..29 en sens inverse faire
    afficher(" ", i)
finPour
→ 29 28 27 26 25 24 23 22 21 20
```

La boucle « ultime » (la plus générale)

Syntaxe

```
boucle
   si <condition 1> alors sortieBoucle finSi
      <instructions 1>
   si <condition 2> alors sortieBoucle finSi
      <instructions 2>
   ...
   si <condition n> alors sortieBoucle finSi
      <instructions n>
   si <condition n+1> alors sortieBoucle finSi
finBoucle
```

Remarques

- `sortieBoucle` (que l'on pourrait appeler `break`) signifie que l'on sort de la boucle.
- Si la seule condition de sortie est `<condition 1>`, la boucle générale se résume à une boucle `tantQue`.
- Si la seule condition de sortie est `<condition n+1>`, la boucle générale se résume à une boucle `faire tantQue`.

Quelle boucle choisir ?

Les planches ci-dessus donnent un peu de culture sur les différentes boucles, ce qui pourra vous servir plus tard.

$\Rightarrow$ En cas d'hésitation entre les différentes boucles, privilégiez la boucle `tantQue`, pertinente dans la très grande majorité des cas.