Notions d'algorithme

Table des matières

1 Introduction

1.1 Algorithme

Définition 1 : Un algorithme est une suite d'instructions, qui une fois exécutée correctement, conduit à un résultat donné.
Pour fonctionner, un algorithme doit donc contenir uniquement des instructions compréhensibles par celui qui devra l'exécuter.

Exemple :

• Voici, ci-contre, un exemple d'algorithme rédigé en langage courant.

Nom : **E1**

Choisir un nombre.
Lui ajouter 1.
Multiplier le résultat par 2.
Soustraire 3 au résultat.
Afficher le résultat.

Si on applique cet algorithme au nombre 3, on a :

$$3 \xrightarrow{+1} 4 \xrightarrow{\times 2} 8 \xrightarrow{-3} 5$$

On peut identifier cet algorithme à une fonction affine :

$$f(x) = 2(x+1) - 3 = 2x + 2 - 3 = 2x - 1$$

• On peut chercher à savoir quel nombre a donné 0 par exemple. Il faut alors remonter l'algorithme, on a alors :

$$0 \xrightarrow{+3} 3 \xrightarrow{\div 2} \frac{3}{2} \xrightarrow{-1} \frac{1}{2}$$

On peut écrire alors le nouvel algorithme ci-contre
La nouvelle fonction affine définie est alors :

$$g(x) = \frac{x+3}{2} - 1 = \frac{1}{2} + \frac{1}{2}$$

Nom : **E'1**

Choisir un nombre.
Lui ajouter 3.
Diviser le résultat par 2.
Soustraire 1 au résultat.
Afficher le résultat.

Remarque : La maîtrise de l'algorithmique requiert deux qualités :

- il faut avoir une certaine **intuition**, car aucune recette ne permet de savoir à priori quelles instructions permettront d'obtenir le résultat voulu.
- il faut être **méthodique et rigoureux**. En effet, chaque fois qu'on écrit une série d'instructions qu'on croit justes, il faut systématiquement se mettre à la place de la machine qui va les exécuter, pour vérifier si le résultat obtenu est bien celui que l'on voulait.

1.2 Conventions pour écrire un algorithme

Historiquement, plusieurs types de notations ont représenté des algorithmes.

- Il y a eu notamment **une représentation graphique**, avec des carrés, des losanges, etc. qu'on appelait des organigrammes. Cependant dès que l'algorithme commence à grossir un peu, ce n'est plus pratique.
- On utilise généralement une série de conventions appelée « **pseudo-code** », qui ressemble à un langage de programmation authentique dont on aurait évacué la plupart des problèmes de syntaxe. Ce pseudo-code est susceptible de varier légèrement d'un livre (ou d'un enseignant) à un autre.

On peut diviser un algorithme en 4 parties : la déclaration des variables, les entrées et l'initialisation, le traitement, les sorties.

Exemple : Les deux algorithmes précédents peuvent s'écrire en pseudo-code comme ci-dessous :

Nom : **E1**

Variables : X réel
Entrées et initialisation
| Lire X
Traitement
| $X+1 \to Y$
| $2Y \to Y$
| $Y-3 \to Y$
Sorties : Afficher Y

Nom : **E'1**

Variables : X réel
Entrées et initialisation
| Lire X
Traitement
| $X+3 \to Y$
| $Y/2 \to Y$
| $Y-1 \to Y$
Sorties : Afficher Y

1.3 Types d'instructions

Les ordinateurs, ne sont capables de comprendre que quatre catégories d'instructions. Ces quatre familles d'instructions sont :

- la lecture / l'affichage
- l'affectation de variables
- les tests
- les boucles

Un algorithmique exprime les instructions résolvant un problème donné indépendamment des particularités de tel ou tel langage de programmation.

Pour écrire un algorithme sur un ordinateur, nous avons besoin d'un langage de programmation. « Un langage de programmation est une convention pour donner des ordres à un ordinateur.»

Il existe des milliers de langage de programmation, ayant chacun ses spécificités. On peut citer par exemple :

Langage	Applications classiques	Compilé/interprété
ADA	Embarqué	compilé
BASIC	Macro de traitement bureautique	interprété
C	Programmation système	compilé
C++	Programmation système objet	compilé
Cobol	Gestion	compilé
Fortran	Calcul	compilé
Java	Programmation orientée Internet	intermédiaire
LISP	Intelligence artificielle	intermédiaire
Pascal	Enseignement	compilé
Prolog	Intelligence artificielle	interprété
Perl	Traitement de chaînes de caractères	interprété

Un **compilateur** est un programme informatique qui transforme un code source écrit dans un langage de programmation (le langage source) en langage machine (le langage cible).
Dans le cas de langage semi-compilé (ou **intermédiaire**), le code source est traduit en un langage intermédiaire, sous forme binaire, avant d'être lui-même interprété ou compilé.

Un **interpréteur** se distingue d'un compilateur par le fait que, pour exécuter un programme, les opérations d'analyse et de traductions sont réalisées à chaque exécution du programme (par un interpréteur) plutôt qu'une fois pour toutes (par un compilateur).

2 Les Instructions

2.1 Création d'un programme

Pour créer un programme, il faut lui donner un nom. Pour la Ti ce nom doit commencer par une lettre et doit contenir au maximum 8 caractères.
Avec la Ti, pour créer un programme faire :

- (prgm) , on sélectionne "NOUV", on valide avec (entrer)
- la calculette est en mode alphanumérique [verr A], on écrit alors le nom désiré et on valide avec (entrer).

2.2 Lecture et affichage d'une variable

a) Définition

Définition 2 : **Lire** une variable signifie que l'utilisateur doit rentrer une valeur pour que le programme la lise

Afficher une variable signifie que le programme renvoie la valeur de la variable que le programme a trouvé.
Ces instructions sont ce qu'on appelle des entrées-sorties, (E/S en français et I/O en anglais)
⚠ Les mots lecture et affichage se situe au niveau du programme

b) Traduction dans le langage de la Ti

Instruction	Langage Ti	manipulations à faire
Lire N	Prompt N	(prgm) E/S 2 : Prompt - puis taper N
Afficher N	Disp N	(prgm) E/S 3 : Disp - puis taper N

2.3 Variable

a) Définition

Définition 3 : Dès que l'on a besoin de stocker une information au cours d'un programme, on utilise une **variable**.

Pour employer une image, une variable est une boîte, que le programme (l'ordinateur) va repérer par une étiquette. Pour avoir accès au contenu de la boîte, il suffit de la désigner par son étiquette.

b) Déclaration des variables

⚠ Avec la calculette Ti, il n'y a pas de déclaration de variable.

La première chose à faire avant de pouvoir utiliser une variable est de créer la boîte et de lui coller une étiquette. C'est ce qu'on appelle la déclaration des variables.

Le nom de la variable (l'étiquette de la boîte) obéit à des impératifs changeant selon les langages. Toutefois, une règle absolue est qu'un nom de variable peut comporter des lettres et des chiffres, mais qu'il exclut la plupart des signes de ponctuation, en particulier les espaces. Un nom de variable correct commence également impérativement par une lettre. Quant au nombre maximal de signes pour un nom de variable, il dépend du langage utilisé.

Une fois le nom choisi, il faut déterminer le type de la variable. On peut distinguer 2 types principaux de variable :

- Type numérique : un nombre (entier, décimal, réel).
- Type alphanumérique : du texte. Dans ce cas pour rentrer une valeur dans cette variable, on met le texte entre guillemets "*texte* "

On peut aussi citer d'autres types de variables
- Type monétaire : un nombre avec deux décimales.
- Type date : jour / mois / année.
- Type booléen : qui ne peut prendre que deux valeurs VRAI ou FAUX.

2.4 Affectation d'une variable

La seule chose qu'on puisse faire avec une variable, c'est l'affecter, c'est-à-dire lui attribuer une valeur.

La flèche d'affectation se trouve pour la Ti sur la touche : sto→

$\boxed{24 \to A}$ Attribue la valeur 24 à la variable a.

$\boxed{A \to B}$ Attribut la valeur de A à la variable B.

$\boxed{\text{"Paul"} \to C}$ Attribut le texte "Paul" à la variable C

Remarque : On peut trouver d'autres notations (plus puristes) pour affecter une valeur à une variable :

$\boxed{A \leftarrow 24}$, $\boxed{A := 24}$ ou $\boxed{A = 24}$ Attribut à la variable A la valeur 24

On peut aussi affecter une variable à l'aide d'une opération :

$\boxed{A + 4 \to C}$ Attribut la valeur $A + 4$ à la variable C.

On peut changer la valeur d'une variable avec elle-même :

$\boxed{B + 1 \to B}$ Augmente de 1 la variable B.

Opérateur alpha numérique : concaniser &

$\boxed{\begin{array}{l}\text{"Paul"} \to A \\ \text{"Milan"} \to B \\ A\&B \to C\end{array}}$ Attribue à le texte "PaulMilan" à la variable C .

Les opérateurs numériques sont : L'addition ⊞, la soustraction ⊟, la multiplication ⊠, la division ⍁, la puissance ⍓

Les opérateurs logiques sont :
- ET les 2 conditions réalisées
- OU (non exclusif) l'une au moins des conditions est réalisée
- NON la condition n'est pas réalisée

3 Les tests

Il y a deux formes pour un test : soit la forme complète, soit la forme simplifiée :

Forme complète

si *condition* **alors**
| instructions 1
sinon
| instructions 2
fin

Forme simplifiée

si *condition* **alors**
| instructions
fin

Si la condition n'est pas vérifiée le programme saute ces instructions.

La condition portant sur une variable peut être :

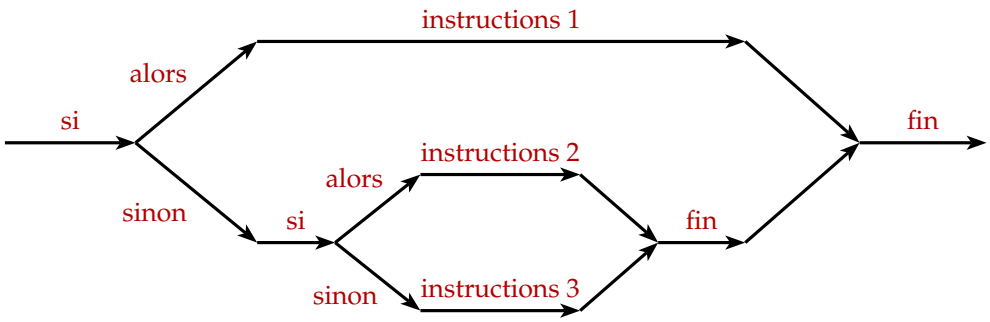
- Une valeur à atteindre.
- Une comparaison avec une autre variable (égalité, inégalité, non égalité)
- Autre valeur

On peut aussi mettre un test qui se décompose en plusieurs conditions reliées par un opérateur logique :
- condition 1 ET condition 2 : les deux conditions doivent être vérifiées en même temps.
- condition 1 OU condition 2 : l'une au moins des deux conditions doivent être vérifiées.

On peut aussi imbriquer un ou plusieurs autres tests à l'intérieur d'un test. On a alors le schéma suivant :

si *condition 1* **alors**
| instructions 1
sinon
| **si** *condition 2* **alors**
| | instructions 2
| **sinon**
| | instructions 3
| **fin**
fin

On pourrait schématiser cette situation par :



Exemple : On donne ci-dessous l'algorithme associée à la valeur absolue, "$abs()$" de votre calculatrice ainsi que sa programmation avec la Ti.

Nom : **VA**

Variables : X, Y réels
Entrées et initialisation
| Lire X
Traitement
| **si** $X \geqslant 0$ **alors**
| | $X \to Y$
| **sinon**
| | $-X \to Y$
| **fin**
Sorties : Afficher Y

```
: Prompt X
: If X ≥ 0
: Then
: X → Y
: Else
: −X → Y
: End
: Disp Y
```

Pour trouver les instructions "If", "Then", "Else", "End" faire dans (prgm) 1 :, 2 :, 3 : ou : 7
Pour trouver le symbole $\geqslant$ faire (2nde) [tests] 4 :
⚠ Les commandes Then et Else sont seuls sur leur ligne

On teste cet algorithme pour -2, 3 et 0. On trouve : 2,3 et 0.

4 Les boucles

4.1 Définition

Définition 4 : Une **boucle** est une structure répétitive ou itérative, c'est à dire que la boucle effectue n fois un calcul sous le contrôle d'une condition d'arrêt.

4.2 La boucle simple

La boucle simple obéit au schéma suivant :

L'instruction Tant que avec la Ti est : **While**
On la trouve en faisant : (prgm) : 5

tant que *condition* **faire**
| instructions
fin

⚠ Dans le cas où la condition est toujours vérifiée, l'ordinateur tourne alors dans la boucle et n'en sort plus. La « **boucle infinie** » est une des hantises les plus redoutées des programmeurs.
Cette faute de programmation est courante chez tout apprenti programmeur.

4.3 Exemple

Soit un algorithme permettant de trouver la partie entière $E(x)$ d'un nombre positif x.
On rappelle que la partie entière $E(x)$ d'un nombre x est définie comme suit :

$$n \leqslant x < n+1 \quad \text{on a :} \quad E(x) = n$$

Nom : **PE**

Variables : N entier, X réel
Entrées et initialisation
  Lire X
  $0 \to N$
Traitement
  tant que $X \geqslant N+1$ **faire**
    $N+1 \to N$
  fin
Sorties : Afficher N

1) Tester cet algorithme avec le nombre $x = 4,3$, en écrivant tous les résultats par boucle.
2) Trouver un algorithme qui permette de calculer la partie entière d'un nombre quelconque (positif ou négatif).

1) La valeur de N au début vaut 0 donc $N+1 = 1$
   1er test $4,3 \geqslant 1$ donc $1 \to N$
   2e test $4,3 \geqslant 2$ donc $2 \to N$
   3e test $4,3 \geqslant 3$ donc $3 \to N$
   4e test $4,3 \geqslant 4$ donc $4 \to N$
   5e test $4,3 \leqslant 5$ donc on affiche 4

2) La définition de la partie entière est la même pour un nombre négatif. Il ne faut pas confondre partie entière et troncature. En effet, la partie entière de $-2,5$ est -3 alors que sa troncature est -2.

Pour écrire un algorithme à partir de l'ancien, on distingue deux cas :

- soit X est positif ou nul, $X \geqslant N+1$ on incrémente alors N de +1
- soit X est négatif, $X < N$ on incrémente N de -1

On obtient alors l'algorithme suivant :

Nom : **PE**

Variables : N entier, X réels
Entrées et initialisation
  Lire X
  $0 \to N$
Traitement
  si $X \geqslant 0$ **alors**
    tant que $X \geqslant N+1$ **faire**
      $N+1 \to N$
    fin
  sinon
    tant que $X < N$ **faire**
      $N-1 \to N$
    fin
  fin
Sorties : Afficher N

4.4 Boucler en comptant

Si l'on connaît à l'avance le nombre de d'incrémentations, on a alors la structure suivante :

pour compteur = initial à final (pas = valeur) **faire**
  instructions
fin

⚠ La valeur du pas est facultative et vaut par défaut 1.

L'instruction Pour avec la Ti est : **For**(compteur, initial, final (,pas)).
On la trouve en faisant : prgm : 4

4.5 Exemple

On considère l'algorithme suivant :

1) Tester, à la main, cet algorithme pour $N = 5$ en donnant les résultats à chaque itération.

2) Pourquoi l'initialisation $1 \to S$ est-elle importante.

3) Écrire cet algorithme avec une calculatrice Ti.

Nom : **FACT**

Variables : N,I, S entiers
Entrées et initialisation
  Lire N
  $1 \to S$ *
Traitement
  pour I *de* 1 *à* N **faire**
    $S \times I \to S$
  fin
Sorties : Afficher S

1) On trouve comme résutat :

I	1	2	3	4	5
S	1	2	6	24	120

2) L'initialisation est important ($S = 1$) car si l'on oublie cette ligne la valeur par défaut de S est 0, ce qui donnera un résultat nul à chaque itération.

3) Voici le programme Ti :

Programme : FACT

: Prompt N
: $1 \to S$
: For$(I, 1, N)$
: $S * I \to S$
: End
: Disp S