Algorithmique
Chapitre 1- Introduction à l'algorithmique

D'après :
Le cours de Stéphanie Laporte
Les présentations de Patrick Reuter

Table des matières

I. Notion de programme

Rappel:

Un ordinateur est une machine électronique programmable servant au traitement de l'information codée sous forme binaire, c'est-à-dire sous forme de tout ou rien (soit le courant passe, soit il ne passe pas).

Contrairement à la vision des films de science-fiction, **un ordinateur est une machine totalement dénuée d'intelligence**. Il n'est capable de traiter qu'un nombre limité d'instructions. Donc il ne faut en aucun cas être intimidé par les ordinateurs: ils sont infiniment plus bêtes que vous. Ce n'est que lorsqu'on réalise vraiment la stupidité des ordinateurs qu'on commence à progresser, car il faut s'abaisser à son niveau: il faut tout lui dire, car il fait tout au pied de la lettre, sans réfléchir.

Pourtant, contrairement aux autres machines qui sont dédiées à un nombre limité de tâches, l'ordinateur est potentiellement capable d'effectuer une infinité de tâches concernant le traitement rationnel de l'information. On dit que c'est une machine **universelle**.

Alors comment une machine stupide peut traiter autant de problèmes différents? C'est que, grâce aux actions de base qu'elle sait réaliser, il est possible en les assemblant de façon pertinente, de résoudre la plupart des problèmes concernant le traitement de l'information. Il suffit de lui indiquer l'ordre dans lequel il faut qu'il effectue ces actions basiques et avec quelles données. Ces ordres élémentaires sont appelés **instructions** et sont rassemblées au sein d'un **programme**.

Comme l'ordinateur a **l'avantage d'exécuter très rapidement et sans erreurs les ordres qu'on lui donne** (les instructions), il exécute beaucoup de traitements complexes plus vite et plus sûrement qu'un homme.

Pour donner des ordres à l'ordinateur, il est nécessaire de pouvoir communiquer avec lui. Cette communication passe par un **langage de programmation**, dans lequel est écrit le programme.

Un programme est un assemblage et un enchaînement d'instructions élémentaires écrit dans un langage de programmation, et exécuté par un ordinateur afin de traiter les données d'un problème et renvoyer un ou plusieurs résultats.

Un algorithme représente l'enchaînement des actions (instructions) nécessaires pour faire exécuter une tâche à un ordinateur(résoudre un problème)
Un algorithme s'écrit le plus souvent en pseudo-langage de programmation (appelé langage algorithmique)

II. Une définition de l'algorithmique ?

Définition Wikipedia (12/9/2005)

> L'algorithmique est la science des algorithmes, visant à étudier les opérations nécessaires à la réalisation d'un calcul.

René Descartes dans le Discours de la Méthode :

> « diviser chacune des difficultés que j'examinerois, en autant de parcelles qu'il se pourroit, et qu'il seroit requis pour les mieux résoudre. ».

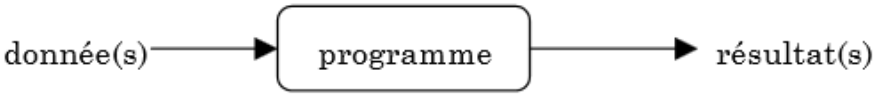
- ✓ La mise en œuvre de l'algorithme consiste en l'écriture de ces opérations dans un langage de programmation et constitue alors la brique de base d'un programme informatique (implémentation, « codage »)
- ✓ L'algorithme devra être plus ou moins détaillé selon le niveau d'abstraction du langage utilisé ; autrement dit, une recette de cuisine doit être plus ou moins détaillée en fonction de l'expérience du cuisinier.

III. De l'art de réfléchir avant de tapoter le clavier

Un algorithme n'est donc exécutable directement par aucune machine. Mais il a l'avantage d'être traduit facilement dans tous les langages de programmation.
L'algorithmique, l'art d'écrire des algorithmes, permet de se focaliser sur la procédure de résolution du problème sans avoir à se soucier des spécificités d'un langage particulier.
Pour résoudre un problème, il est vivement conseillé de réfléchir d'abord à l'algorithme avant de programmer proprement dit, c'est à dire d'écrire le programme en langage de programmation.

IV. Notion de variables

Les programmes ont pour but de traiter différentes données afin de produire des résultats. Les résultats peuvent eux-mêmes être des données pour d'autres programmes.



Les données d'un programme doivent être récupérées en mémoire centrale, à partir du clavier ou d'un fichier par exemple, pour pouvoir être traitées par le processeur qui exécute le programme. Ainsi, toutes les données d'un programme sont mémorisées en mémoire centrale, dans des sortes de cases que l'on appelle variables.

Une variable peut être représentée par une case mémoire, qui contient la valeur d'une donnée.

Chaque variable possède un nom unique appelé identificateur par lequel on peut accéder à son contenu.

Par exemple, on peut avoir en mémoire une variable prix et une variables quantité qui contiennent les valeurs 10.2 et 5

Attention à ne pas confondre la variable et son contenu

Une variable est un contenant, c'est à dire une sorte de boîte,

alors que le contenu d'une variable est une valeur numérique, alphanumérique ou booléenne, ou de tout autre type

Deux variables peuvent avoir la même valeur, **mais une variable ne peut pas avoir plusieurs valeurs en même temps.**

En revanche, la valeur d'une variable peut varier au cours du programme. L'ancienne valeur est tout simplement écrasée et remplacée par la nouvelle.

Les variables dont la valeur ne change pas au cours de l'exécution du programme sont appelées variables constantes ou plus simplement constantes.

V. L'instruction qui associe une variable et une valeur : L'affectation

L'affectation consiste tout simplement à placer une valeur dans une variable (ce qui revient à changer le contenu de cette variable)

La nouvelle valeur est évaluée à partir d'une expression, qui peut être
- soit une autre variable ou constante,
- soit une valeur littérale
- soit une combinaison de variables, de valeurs littérales et d'opérateurs

☞ Exemple:
Supposons que *rayon* soit une variable de valeur 5
2 * *rayon* * 3.14 est une expression qui vaut 31.4

En algorithmique, pour représenter l'opération d'affectation, on va utiliser le symbole ←. On évite d'utiliser le symbole = pour ne pas le confondre avec l'égalité mathématique.

x ← 5 ;
x ← y ;
x ← 5 + 8 ;
x ← 5 + y ;

Les instructions d'affectation sont formées de deux parties :

- **A gauche du symbole ←, on trouve toujours le nom d'une variable** destinée à recevoir une valeur.
- A droite, on trouve l'expression qui va donner la valeur qu'on veut affecter à la variable en question.

L'instruction d'affectation agit en deux temps :

- Tout d'abord elle détermine la valeur de l'expression à droite du ← (la plupart du temps, il n'y a même pas de calculs à faire!)
- puis elle range la résultat dans la variable située à gauche.

Complément concernant l'affectation de texte :

Les valeurs numériques, nous l'avons vu, s'écrivent de façon naturelle, sans avoir à les accompagner de symboles supplémentaires. Il n'en est pas de même avec les caractères et les chaînes, qu'il faut entourer de guillemets ou d'apostrophe. Sinon, en effet, le programme pourrait confondre le caractère ou la chaîne avec le nom d'une variable. Les chaînes sont entourées de guillemets et les caractères de simples apostrophes.

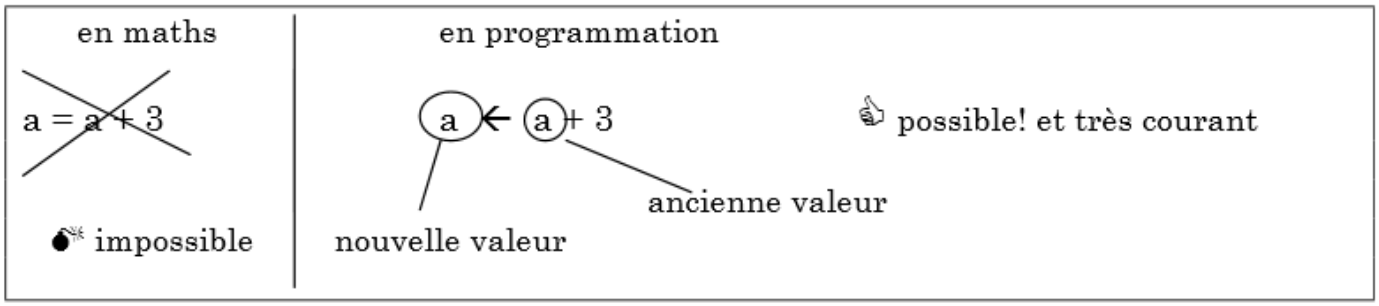
Exemples:
réponse ← 'o'
mot ← « bonjour »

VI. L'affectation est bien différente de l'égalité mathématique

Quand on dit qu'une variable prend pour valeur une autre variable, ça ne veut pas dire qu'elles seront toujours égales! Cela veut seulement dire qu'au moment de l'instruction, la première variable va prendre la valeur de la seconde.

Voici les principales différences entre l'affectation et l'égalité mathématique:

- L'instruction x ← y n'a pas la même signification que l'instruction y ← x contrairement aux mathématiques où x = y est équivalent à y = x. L'instruction x ← y signifie que x va prendre la valeur de y. Au contraire y ← x signifie que y va prendre la valeur de x.
- On ne peut affecter de valeur qu'à une variable. Il est impossible d'affecter une valeur à une expression. Ainsi, par exemple, l'instruction a + 5 ← b n'a pas de sens, alors que a + 5 = b a une signification en mathématiques.
- Il est possible de retrouver la même variable à droite et à gauche de l'expression. Ecrire a ← a + 3 a un sens en programmation, alors que a = a + 3 n'a pas de sens en mathématiques. Cela signifie que l'on évalue a + 3 avec l'ancienne valeur de a et qu'on range le résultat dans a. La valeur de a sera donc augmentée de 3 par cette opération.



Exercice 1 : Possible ou impossible ?

a <- 7
score <- 0
score <- score + 100
gameover <- FAUX
a+b <- 6
7 <- c

Exercice 2 : Quelle valeur dans chaque variable après chaque instruction ?

A ← 5
B ← 3
C ← A + B
A ← 2
C ← B - A

VII. Peut-on faire tout et n'importe quoi avec ces variables ? Évidemment non, déclaration des variables (même si SCILAB ne nous y oblige pas)

Pour qu'un programme puisse utiliser une variable, il faut au préalable que cette variable ait été déclarée, c'est-à-dire que le programme lui ait réservé une place en mémoire et ait attribué l'identificateur à cette place.

Mais toutes les variables n'ont pas besoin de la même place en mémoire. Un grand nombre prend plus de place qu'un caractère. Selon le type de l'objet, il faudra lui réserver plus ou moins de place: c'est pourquoi il faut déclarer le type des variables et pas seulement leur nom. Par ailleurs, selon le type des variables, les opérations possibles seront différentes.

Donc la déclaration d'une variable indique deux choses:
- son **identificateur** (son nom)
- son **type** (sa taille)

Un identificateur peut être composé de lettres et de chiffres mais il ne peut pas commencer par un chiffre et ne peut comporter d'espaces.
L'identificateur des variables doit être suffisamment signifiant pour qu'on reconnaisse leur fonction aisément. Par exemple pour des variables représentant un prix et une quantité, évitez *a* et *b* mais utilisez plutôt *prix* et *quant*.

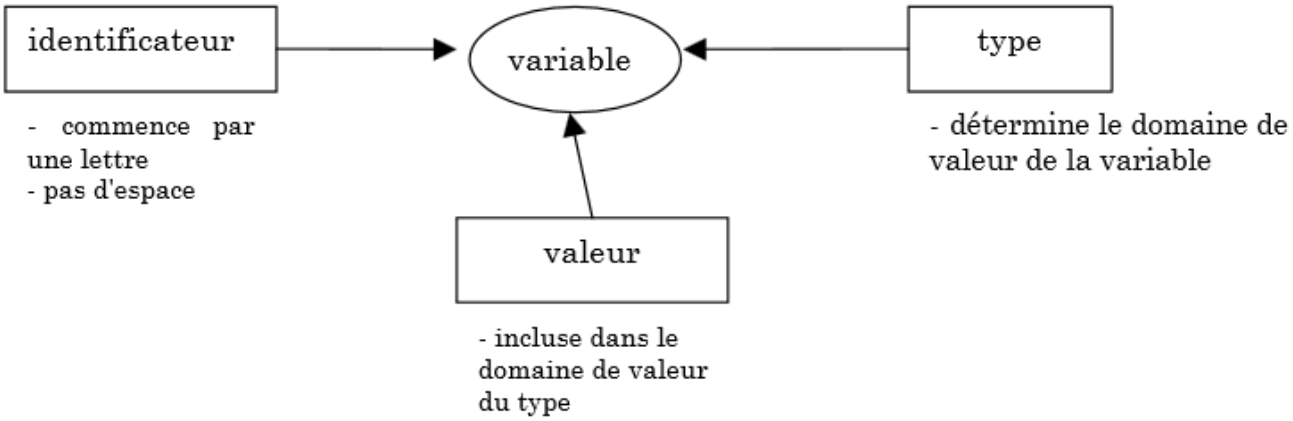
En algorithmique, on distingue 5 types principaux:

- les **caractères** (lettres, chiffres, ponctuation, code des opérations, espace, retour chariot,… et plus généralement toutes les touches que l'on peut trouver sur une machine à écrire)
- les **chaînes** de caractère (suites de caractères)
- les **entiers** (les nombres sans virgule)
- les **réels** (les nombres à virgule et sans virgule)
- les **booléens** (qui n'ont que deux valeurs possibles: soit VRAI, soit FAUX)

[A1]

Les opérations possibles sur les variables dépendent de leur type (voir page suivante)

Synthèse:



1-dans certains langages, il est possible de ne pas déclarer le type des variables. Ce sont des langages faiblement typés. Mais ceci n'est pas recommandé, c'est pourquoi en algorithmique, nous prendrons l'habitude de toujours déclarer le type des variables

VIII. Quelles opérations possibles pour quel type de données ?

Type	Exemple	opérations possibles	symbole ou mot clé correspondant
réel	-15.69 , 0.36	addition soustraction multiplication division exposant pourcentage comparaisons	+ – * (et pas x pour ne pas confondre avec la lettre x) / ^ % <, ≤,>, ≥, =, ≠
entier	-10, 3, 689	addition soustraction multiplication division modulo exposant pourcentage comparaisons	+ – * (et pas x pour ne pas confondre avec la lettre x) DIV cf. ci-après ① MOD cf. ci-après ② ^ % <, ≤,>, ≥, =, ≠
caractère	'B' 'h' '£' '?' '\n'	comparaisons	<, ≤,>, ≥, =, ≠ cf. ci après ③
chaine	"Bonjour" "93000" "toto@caramail.com"	concaténation longueur extraction	& cf. ci-après ④ longueur(*chaîne*)
booléen	VRAI, FAUX	comparaison négation conjonction disjonction	<, ≤,>, ≥, =, ≠ NON ET OU

① Pour les entiers, la division est notée **Div**. Elle est nommée division entière et diffère un peu de la division que l'on trouve sur les calculettes. Elle ne donne que le chiffre avant la virgule du résultat (elle renvoie un entier).

② Les entiers supportent une opération supplémentaire appelée **modulo**, notée **mod** et qui renvoie le reste de la division entière.

Exemple:

7 / 2 donne 3.5
7 Div 2 donne 3
7 Mod 2 donne 1

③ Les caractères sont comparés selon l'ordre du code ASCII. C'est ainsi qu'on peut comparer tous les caractères entre eux. Par exemple la lettre Z (majuscule), de code ASCII 90 est inférieure à la lettre a (minuscule) de code ASCII 97. L'ordre ASCII des lettres de la même casse suit l'ordre alphabétique, de sorte que A<B<C<D<…

④ L'opérateur & sert à concaténer des chaînes de caractère, ce qui signifie transformer plusieurs chaînes en une seule en les ajoutant les unes à la suite des autres.
Ex : « Bonjour » & « à tous » donne « Bonjour à tous »

IX. Les figures imposées d'un programme et donc d'un algorithme (même si SCILAB ne nous y oblige pas)

```
PROGRAMME toto

/* les constantes:
 il est obligatoire de leur donner une valeur dès leur déclaration */
CONST     titi ← 10 : entier
          tutu ← "bonjour!" : chaîne

// les variables au sens strict
VAR       riri, fifi : réels
          loulou : chaîne

DEBUT

/* instructions*/

FIN
```

déclarations

corps du programme

Les mots du langage algorithmique sont écrits en gras ou soulignés. Un algorithme commence par le mot PROGRAMME suivi de son identificateur (le nom du programme).

Ensuite viennent les déclarations: dans un premier temps celles des constantes, annoncée par CONST, puis celle des variables, annoncée par VAR. Pour déclarer une variable, on indique son identificateur suivi d'un double point et de son type. La valeur des constantes est donnée dès leur déclaration, avec le signe ← précédé de son identificateur.

Le corps du programme commence par DEBUT et se termine par FIN. On peut insérer des commentaires, soit entre les balises /* */, soit après // jusqu'à la fin de la ligne.

L'exécution d'un programme est constituée :
- d'échanges d'informations en mémoire
- de calculs

Une instruction est un ordre élémentaire que peut exécuter directement l'ordinateur. Une instruction revient à déplacer une information d'un endroit à un autre de la mémoire.

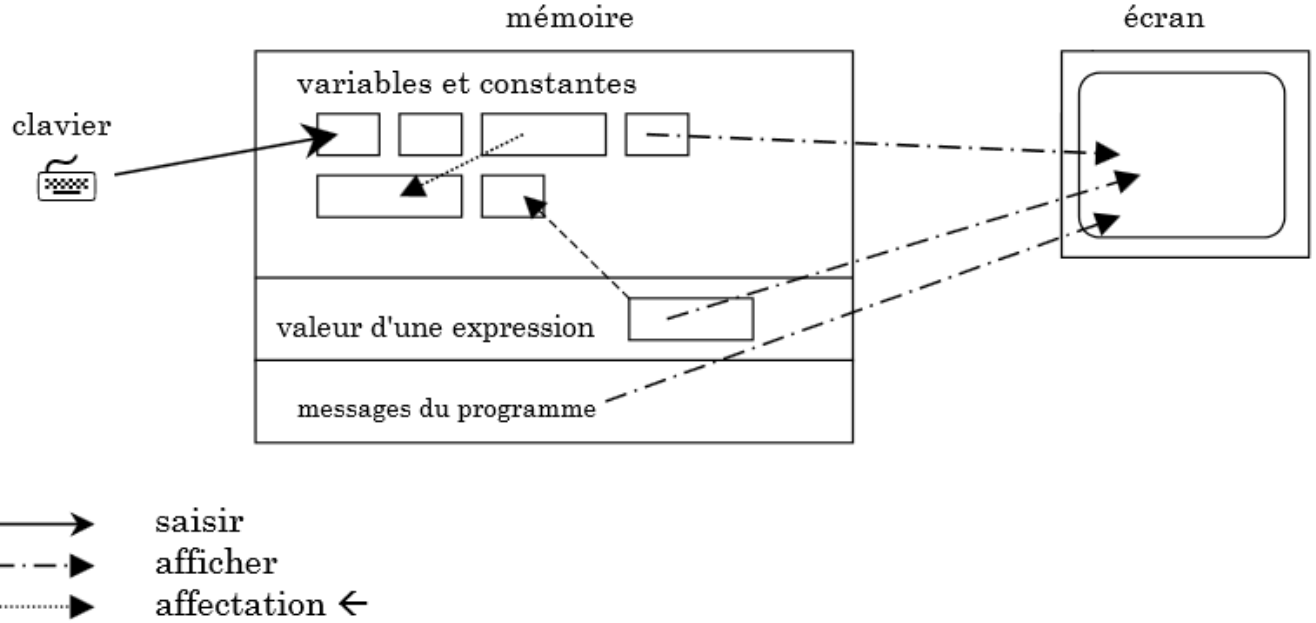
Les informations (données) manipulées par les instructions peuvent prendre plusieurs formes:
- des **variables** proprement dites
- des variables **constantes**
- des **valeurs littérales** (écrites telles qu'elles dans le programme: ex "bonjour", 45, VRAI)
- des **messages** (libellés) envoyés à l'utilisateur (quelles données sont à entrer, quels résultats sont affichés…), qui sont des valeurs littérales particulières
- des **expressions** complexes (combinaisons de variables, constantes et valeurs littérales avec des opérateurs) ex : 2 * r * 3.14

Les instructions élémentaires les plus courantes sont :

- l'**affectation: le fait de donner une nouvelle valeur à une variable**
- l'**affichage** sur l'écran
- la **saisie** à travers le clavier

D'autre instructions permettent de lire et d'écrire sur d'autres périphériques: nous les étudierons plus tard.

Illustration des instructions élémentaires



Exemple :

Programme test

```
VAR
a, b : entiers
//a et b ont une valeur indéterminée

DEBUT
  a ← 3                  // a vaut 3
  // b n'a pas encore de valeur (ou plus exactement a une valeur indéterminée)
  b ← a + 2    // b vaut 5, a vaut toujours 3
  a ← b * 2    // a vaut 10, b vaut 5
  b ← b + 1    // b vaut 6
FIN
```

X. L'initialisation des variables : après leur déclaration et avant leur affectation

Au début d'un programme, les variables n'ont pas encore reçues de valeur ; on dit qu'elles sont **indéfinies**. Toutes les variables doivent être initialisées, c'est-à-dire recevoir une valeur initiale, avant leur utilisation (à droite d'une affectation ou dans une expression conditionnelle). Sinon, le contenu de la variable n'est pas vide, il n'est pas forcement nul, il est quelconque. Ainsi, lorsqu'on utilise une variable indéfinie (qui n'a pas encore reçue de valeur), le comportement du programme va être aléatoire.

De là vient la nécessité de toujours bien initialiser les variables avant d'utiliser leur valeur.[1] L'initialisation s'effectue généralement au début du programme, juste après les déclarations. Elle prend la forme d'une simple **affectation ou** d'une **saisie au clavier**.

```
Programme initialisation
/*déclarations*/
VAR      x , y  : entiers
         a : chaîne

DEBUT
/*initialisation*/
x ← 0
y ← 10
a ← "hello"
/* autres instructions */
FIN
```

[1] Notez que la saisie d'une variable au clavier permet, comme l'initialisation d'affecter une valeur initiale à une variable: donc les variables saisies n'ont pas à être initialisées.

XI. Instruction en input : la saisie

L'instruction de saisie permet de communiquer des données au programme. Cette instruction assigne une valeur entrée au clavier dans une variable. Tant que l'utilisateur n'entre rien au clavier, le déroulement du programme est stoppé.

Saisir *variable1 [, variables2, ...]**

☞ Exemples:

Saisir x
Cette instruction va lire la valeur saisie au clavier et l'affecte à la variable x

Saisir x, y
Cette instruction lit la première valeur saisie au clavier et l'affecte à x, puis lit la deuxième valeur saisie et l'affecte à y

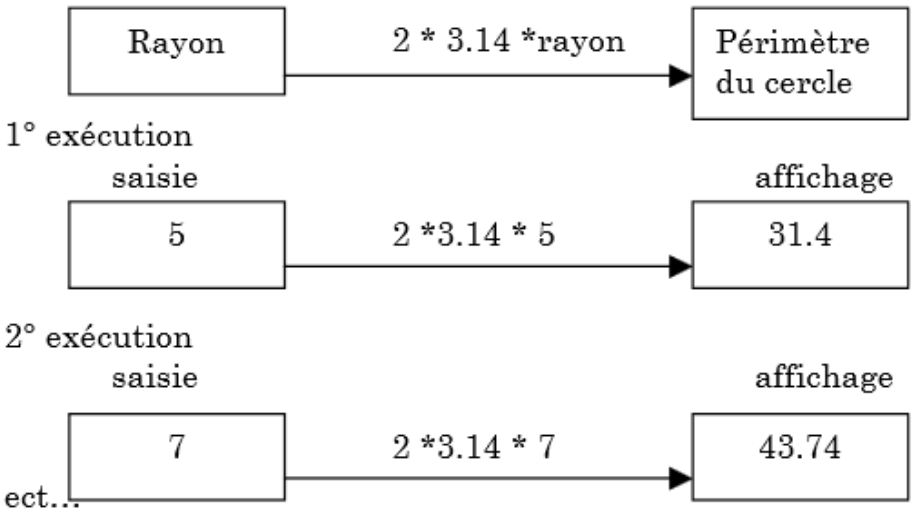
➢ Utilité de la saisie

On pourrait se dire que l'instruction de saisie est inutile car on dispose déjà un moyen d'attribuer une valeur aux variables, par l'instruction d'affectation. Mais en fait, l'instruction de saisie (ou de lecture sur un périphérique autre que le clavier) est indispensable pour permettre d'utiliser le même programme sur des données différentes sans avoir à changer les valeurs du programme à chaque fois.

Par exemple, l'instruction lire x à laquelle on fournirait 5, pourrait être remplacée par x ← 5. Mais alors si on veut utiliser le programme avec une autre valeur, il faudra aller changer le programme. En revanche, si on utilise une instruction de saisie, le choix de la valeur se fait en cours d'exécution du programme. On peut donc utiliser le programme autant de fois que l'on veut avec des données différentes sans avoir à modifier le programme.

☞ Exemple:

Le traitement qui consiste à calculer le périmètre d'un cercle à partir de son rayon peut s'effectuer pour toutes les valeurs possible de rayon. A chaque exécution du programme, on fournit une valeur différente au rayon, ce qui va donner un périmètre différent.



Sans instruction de saisie (ou de lecture sur un périphérique quelconque), un programme fournirait toujours le même résultat.

XII. Instruction en output : l'affichage

La plupart des programmes nécessitent de communiquer à l'utilisateur un certain nombre de résultats par l'intermédiaire d'un périphérique. Pour cela, ils utilisent des instructions d'affichage.

L'instruction d'affichage permet de fournir des résultats sous forme directement compréhensible pour l'utilisateur à travers l'écran.

➢ Syntaxe

Afficher *expression1, [expression2]*

☞ Exemples

Afficher toto

Cette instruction permet d'afficher la valeur de la variable toto à l'écran
Si toto est une chaîne qui vaut "tutu", cette instruction affichera tutu à l'écran

Afficher "Bonjour!"

Celle-ci permet d'afficher la chaîne littérale Bonjour! à l'écran

Afficher a, b

Quand on veut afficher deux objets à la suite, on les sépare d'une virgule
Si a vaut 5 et b vaut 10, on obtient alors à l'écran:

```
5 10
```

✍ Remarque: dans la plupart des langages (dont le C++), il faut ajouter dans le programme un espace entre les différentes expressions à afficher, mais en algorithmique on ne s'embête avec ces détails.

On peut mélanger l'affichage de valeur littérales et de variables. Cela est particulièrement utile si on veut voir apparaître un **libellé (texte accompagnant la saisie des données ou l'édition des résultats, permettant de guider l'utilisateur)**.

☞ Exemple

Afficher « Voici les résultats : x = », x, « et y = », y
Ils apparaîtront dans l'ordre. Nous aurons donc à l'écran (en supposant que les valeurs de x et y sont respectivement 5 et 10) :

```
Voici les résultats : x = 5 et y = 10
```

Exercice 3 : Possible ou impossible ?

Compteur <- 8 ; Nom <- "Patrick"; 7+3 <- 10 ; resultat <- 20*5 ;	20 <- c ; score <- score + 10; highscore <- score;

Exercice 4 : Quelle valeur dans chaque variable après chaque instruction ?

A ← 5
B ← A + 4
A ← A + 1
B ← A - 4

Exercice 5 : permutation de nombres [A2]

Ecrivez un algorithme qui demande la valeur de 2 variables texte a et b et les permute ? (Élément de réponse : si on commence par écrire a = b on va détruire la valeur de a ... Il va falloir utiliser une troisième variable...)

Exercice 6 : Comparez les deux algorithmes suivants, qu'affichent-ils ?

```
Programme Somme1
// déclaration des constantes
CONST
    Nombre1 ←234 : entier
    Nombre2 ←166 : entier
// déclaration des variables
VAR
    Résultat : entier
//corps du programme
DEBUT
// traitement
    Résultat ← Nombre1 + Nombre2
//Affichage du résultat
    Afficher "la somme de ",Nombre1," et de
    ",Nombre2," est ",résultat
FIN
```

```
Programme Somme2
// déclaration des variables
VAR
    Nombre1, Nombre2,Résultat : entier
//corps du programme
DEBUT
// traitement
    Afficher "veuillez saisir le Nombre1"
    Saisir Nombre1
    Saisir "Veuillez saisir Le Nombre2",Nombre2
    Résultat ← Nombre1 + Nombre2
//Affichage du résultat
    Afficher "la somme de ",Nombre1," et de
    ",Nombre2," est ",résultat
FIN
```

Exercice 7 : carré d'un nombre

Ecrivez un algorithme qui demande un nombre `à l'utilisateur, puis qui calcule et affiche le carré de ce nombre.

Exercice 8 : TTC

Ecrivez un algorithme qui lit le prix HT d'un article, le nombre d'articles et le taux de TVA, et qui fournit le CA TTC correspondant. Faire en sorte que des libellés apparaissent clairement.

Exercice 9 : Calcul de l'hypoténuse

Calculez l'hypoténuse d'un triangle connaissant les deux côtés de l'angle droit

Exercice 10 : Conversion de temps

A partir d'un temps donné en secondes, convertissez-le en heures, minutes et secondes
Indications pour réaliser cet algorithme sans utiliser la structure SI…ALORS…SINON…FINSI :

Mettre dans q le quotient de t par 60 (nombre de minutes)
Mettre dans s le reste (nombre de secondes)
Mettre dans h le quotient de q par 60 (nombre d'heures)
Mettre dans m le reste (minutes restantes)