Chapitre 2 : Notions d'algorithme et de programme

Partie 1

1. Concept d'un algorithme

L'algorithme est une description d'un traitement automatisé de données destiné à être réalisé sur un ordinateur, après avoir traduit cette description dans un langage de programmation. (Description en langage naturel de la suite des actions effectuées par un programme) (Description d'une suite d'actions à effectuer, dans un ordre donné, pour parvenir un résultat) (Suite finie d'instructions permettant de donner la réponse à un problème)

Exemple :

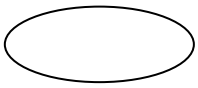
1. Ouvrir la porte du garage
2. Prendre la clef
3. Ouvrir la porte avant gauche
4. Entrer dans la voiture
5. Mettre au point mort
6. Mettre le contact
7. Débrayer
8. Enclencher la marche arrière
9. Desserrer le frein à main
10. Embrayer doucement
11. …

Pour résoudre un sujet d'examen *:*

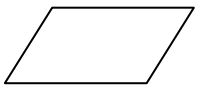
1. *Lire tout le sujet*
2. *Choisir l'exo le plus facile*
3. *Cas de Pb, passer à l'exo suivant jusqu'à la fin*
4. *Puis revenir aux autres exo et leurs donner le tp qu'il faut*
5. *Fin*

- *Recette culinaire*
- *Montrer le chemin à quelcun*
- *Résoudre une équation du $2^{ème}$ degré*

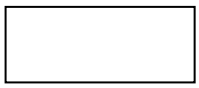
2. Représentation en organigramme : c'est la traduction graphique de l'algorithme.



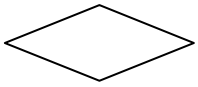
Début et fin



E/S (Ecriture/Lecture)



Action (ou instruction)



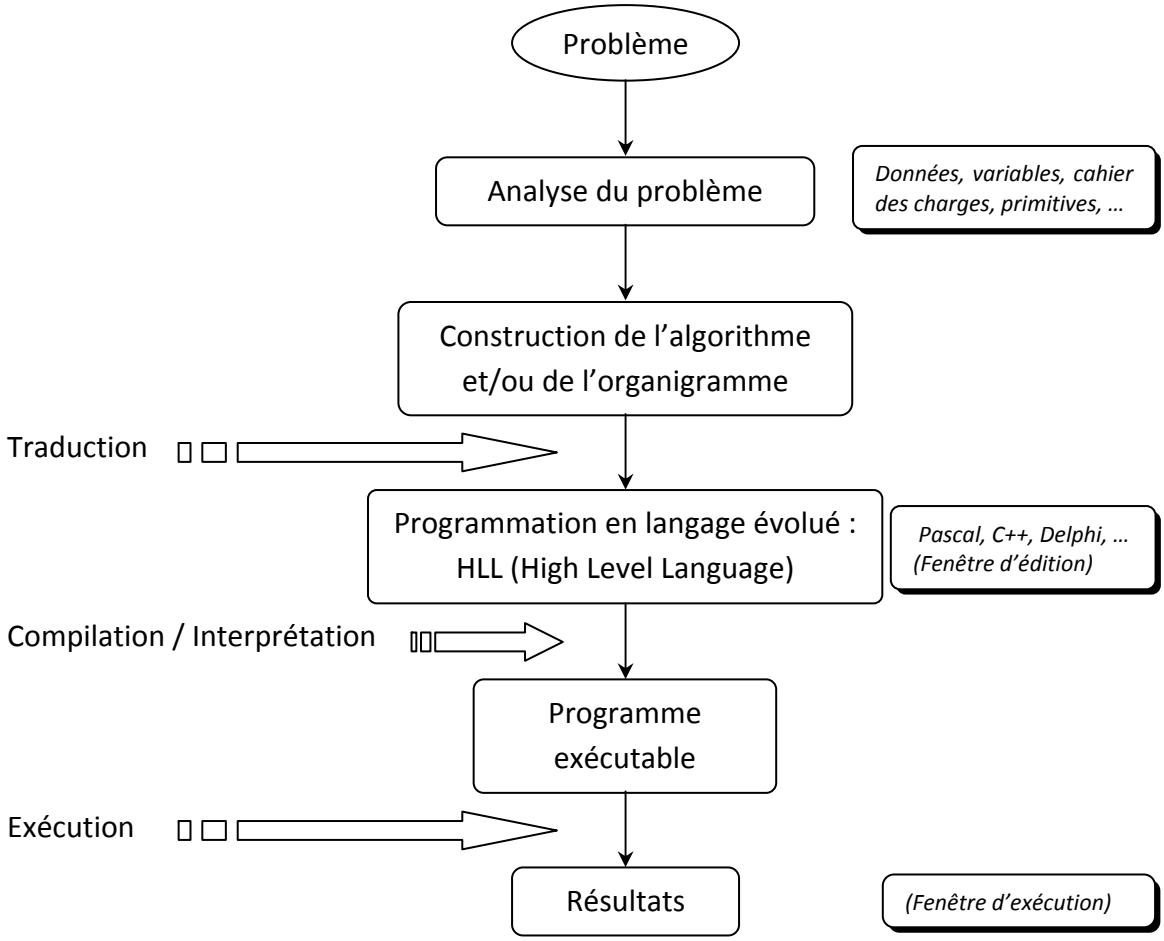
Condition (ou question) (test)

Direction (sens)

Il est plus clair que l'algo, mais dès que l'algo se complique, il devient très difficile et même impossible.
Il est parfois appelé Algorigramme ou Ordinogramme

3. Démarche et analyse d'un problème

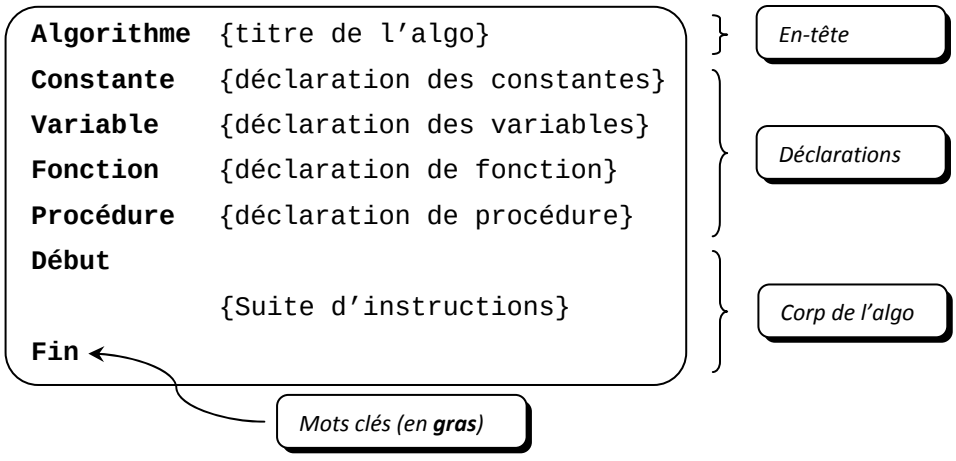
Pour résoudre un problème par ordinateur, il faut passer par les étapes suivantes :



Résolution d'un problème par ordinateur

4. Structure d'un programme

Voici la structure générale d'un algorithme :



Et voici celle d'un programme Pascal :

```
Program     {nom du programme} ;
Uses        {unités (ou bibliothèques) utilisées} ;
Const       {déclarations des constantes} ;
Type        {déclaration des types} ;
Function    {déclaration de fonction} ;
Procedure   {déclaration de procédure paramétrée} ;
Var         (* déclaration des variables *) ;
Procedure   (* déclaration de procédure simple *) ;
Begin
            {Instructions du programme principal} ;
End.
```

NB : L'ordre indiqué doit être impérativement respecté.

- **Uses** : Indique qu'on utilise des fonctions et outils contenus dans d'autres fichiers de programme (nommés des unités) que celui dans lequel on travaille.
- **Type** : Cette clause sert à définir des types de données particulières (par exemple : les jours de la semaine, les couleurs, …).

*Attention : Le mot réservé **Type** ne doit être écrit qu'une seule fois.*

*Par exemple : Pour pouvoir utiliser la procédure « **Clrscr** » mettre en début de programme la commande : **Uses crt** ; Sous Windows : **Uses wincrt** ;*
***Clrscr** : procédure sans paramètre ; efface le contenu de l'écran et positionne le curseur au coin supérieur gauche.*
***Procedure** et **Function**: On verra plus tard. L'essentiel est de savoir que ces procédures et fonctions sont situées AVANT le bloc principal **Begin/End**.*

Grammaire du Pascal :

- Un identificateur (nom d'une constante, variable, fonction, programme, …) est un mot composé de lettres, de chiffres et du caractère « _ » et commençant obligatoirement par une lettre.
- Un identificateur ne peut être égal à un mot réservé du langage **Pascal** ! Les mots réservés (mots clés) varient d'un compilateur à l'autre (**Turbo Pascal, Free Pascal**, **GnuPascal**, **Delphi**, . . .) et d'une version à l'autre.
- Les identificateurs ne doivent pas excéder **127** signes (selon compilateur) (1 lettre au minimum).

Exemple : LMD, D45, ST_2, aaa1.

- **Turbo Pascal** ne différencie aucunement les majuscules des minuscules. Ainsi les identificateurs *Ma_Variable* et *ma_variable* sont considérés comme équivalents.
- Un programme principal débute toujours par **Begin** et se termine par **End.** (avec un point). Alors qu'un sous-programme (fonction, procédure, bloc conditionnel...) commence lui aussi par **Begin** mais se termine par **End ;** (sans point mais avec un *point-virgule*).

- Chaque commande doit se terminer avec un *point-virgule* (**;**). Il n'y a pas d'exception à la règle hormis **Begin** et l'instruction précédent **End** ou **Else**.
- Il est toléré de mettre plusieurs instructions les unes à la suite des autres sur une même ligne du fichier mais il est recommandé de n'en écrire qu'une par ligne : c'est plus clair et en cas de bogue, on s'y retrouve plus aisément. De plus, s'il vous arrive d'écrire une ligne trop longue, le compilateur vous le signifiera en l'erreur « Error 11: Line too long ». Il vous faudra alors effectuer des retours à la ligne comme le montre l'exemple suivant :

```
WriteLn('Fichier: ', file,
' Date de création:', datecrea,
' Utilisateur courant:', nom,
' Numéro de code:', Round(ArcTan(x_enter)*y_old):0:10) ;
```

- Les commentaires sont des éléments du texte d'un programme qui sont ignorés par le compilateur. Ils servent à apporter des renseignements sur le programme.

  En **Pascal**, il y a trois façons d'écrire des commentaires :

  1. commentaires sur une seule ligne : ils débutent par **//**

     ```
     // Commentaire sur une s e u l e   l i g n e
     ```

  2. commentaires sur plusieurs lignes : encadrés par **{** et **}**

     ```
     { commentaire sur
     p l u s i e u r s
     l i g n e s }
     ```

  3. commentaires sur plusieurs lignes : encadrés par **(*** et ***)**

     ```
     (* commentaires
     sur p l u s i e u r s
     l i g n e s *)
     ```

N'hésitez pas à insérer des commentaires dans votre code, cela vous permettra de comprendre vos programmes un an après les avoir écrit, et ainsi d'autres personnes n'auront aucun mal à réutiliser vos procédures, fonctions ... Vous pouvez aussi mettre en commentaire une partie de votre programme.

5. Structure des données

5.1. Types de données

Un programme peut être amené à manipuler plusieurs types de données :

Types simples (de base) :

- ***Booléen*** : Valeur pouvant être soit « Vraie » soit « Fausse ».
- ***Entier*** : Valeur numérique entière pouvant être signée ou non signée. (5,-20, …)
- ***Réel*** : Valeur numérique codée avec Mantisse et Exposant ($M.10^E$). ($1,3.10^{20}$, $1,6.10^{-19}$, …)
- ***Caractère*** : Octet correspondant à un code ASCII. ("A", "m", "@", "5", …)
- ***Chaînes de caractères*** : Ensemble de caractères. ("Mohammed", "Salam", …)

Types composés (complexes)

Tableau de données : Ensemble fini de données de même type. (Matrices)

(2, 7, 11)

$\begin{pmatrix} 1,3 \\ 3,8 \\ 7,2 \end{pmatrix}$

"a"	"#"
"4"	"Z"

Enregistrement (Structure) : Ensemble de données de types différents correspondant à un même objet. (Etudiant : Nom, Prénom, Age, Filière, Adresse, Moyenne, Admi, …)

Les types les plus utilisés en **Pascal** sont :

- ***Integer*** (nombres entiers).
- ***Real*** (nombres réels).
- ***Char*** (caractère) permet de représenter les caractères. Les constantes de type caractère s'écrivent entre apostrophe :'A', '3' ou '*'.
- ***String*** (chaînes de caractères).
- ***Boolean*** prend deux valeurs possibles: *TRUE* et *FALSE*.

Petite liste-exemple très loin d'être exhaustive : (Annexe)

Désignation	Description	Bornes	Place en mémoire
Real	nombres réels (avec 11 décimales)	$2.9\ 10^{-39}$ **et** $1.7\ 10^{+38}$	6 octets
Single	réel	$1.5\ 10^{-45}$ **et** $3.4\ 10^{+38}$	4 octets
Double	réel (avec 15 décimales)	$5.0\ 10^{-324}$ **et** $1.7\ 10^{+308}$	8 octets
Extended	réel	$1.9\ 10^{-4951}$ **et** $1.1\ 10^{+4932}$	10 octets
Comp	réel	$-2\ 10^{+63}+1$ **et** $2\ 10^{+63}+1$	8 octets
Integer	nombres entiers (sans virgule)	-32 768 **et** 32 767	2 octets
Longint	entier	-2 147 483 648 **et** 2 147 483 647	4 octets
Shortint	entier	-128 **et** 127	1 octet
Word	entier	0 **et** 65 535	2 octets
Byte	entier	0 **et** 255	1 octet
Long	entier	$(-2)^{31}$ **et** $(2^{31})-1$	4 octets
Boolean	variable booléenne (valeurs logiques)	TRUE **ou** FALSE	1 octet
Array [1..10] Of xxx	tableau de 10 colonnes fait d'éléments de l'ensemble défini xxx (Char, Integer...)		
Array [1..10, 1..50, 1..13] Of xxx	tableau en 3 dimensions fait d'éléments de l'ensemble défini xxx (Char, Integer...)		
String	chaîne de caractères		256 octets
String [y]	chaîne de caractère ne devant pas excéder y caractères		y+1 octets
Text	fichier texte		
File	fichier		
File Of xxx	fichier contenant des données de type xxx (Real, Byte...)		
Char	nombre correspondant à un caractère ASCII codé	0 et 255	1 octet
Pointeur	adresse mémoire		4 octets
Datetime	format de date		
Pathstr	chaîne de caractère (nom complet de fichier)		
Dirstr	chaîne de caractère (chemin de fichier)		
Namestr	chaîne de caractère (nom de fichier)		
Extstr	chaîne de caractère (extension de fichier)		

5.2. Constantes et variables

a. Constante : Donnée manipulée par un programme et ne pouvant pas être modifiée.

Les constantes se déclarent comme suit :

En **algorithmique** :

```
Constante    <Nom_Constante> = <Valeur>
```

En **Pascal** :

```
Const    <Identificateur> = <Valeur> ;
```

L'utilisation de constantes en programmation est vivement conseillée. Elles permettent :

- une notation plus simple (*Exemple* **: Pi** à la place de **3.141592653**).
- la possibilité de modifier simplement la valeur spécifiée dans la déclaration au lieu d'en rechercher les occurrences, puis de modifier dans tout le programme.

Exemple :

```
Const       pi = 3.141592653 ;                    {type numérique}
            DEUX = 2 ;
            VRAI = true;                          {type booléen}
            FAUX = false;
            CAR_A  = 'A';                         {type caractère}
            PHRASE = 'Salamo 3alikom';    {type chaîne de caractères}
            LARG = 640 ;
            HAUT = 480 ;
            NB_PIX = LARG*HAUT ;
```

Il est possible de déclarer plusieurs constantes.
NB : Ne pas oublier le point virgule à la fin de chaque déclaration.

b. Variable : Donnée manipulée par un programme et pouvant être modifiée. Ce peut être :

- Une donnée d'entrée (exemple : A, B, C)
- Un résultat final de calcul (exemple : X)
- Un résultat intermédiaire de calcul (exemple : Δ)

Equation du 2ème degré : ***A.X^2+B.X+C=0***

Déclarer une variable c.-à-d. ; Allouer un espace mémoire (réserver une case mémoire et lui donner un nom).

Les variables se déclarent comme suit :

En **algorithmique** :

```
Variable    <Nom_Variable> : <Type>
```

En **Pascal** :

```
Var    <Identificateur> : <Type> ;
```

NB :
- ***Var*** *ne peut apparaître qu'une seule fois.*
- *Possibilité de grouper plusieurs variables pour le même type (séparation des variables par une virgule).*

Exemple :

```
Var
      x : Real ;
      i, j, n : Integer ;
      FLAG : Boolean ;
      s , t : String ;
      C1 , C2 : Char ;
```

6. Les opérateurs

6.1. Les opérateurs arithmétiques

Addition	**+**
Soustraction	**-**
Multiplication	*****
Division réelle	**/**

Div : renvoie le quotient de la division entière x **Div** y
Mod : renvoie le reste de la division entière x **Div** y

- *Pour les entiers : **+, -, *, Div, Mod,***
- *Pour les réels : **+, -, *, /,***

6.2. Les opérateurs relationnels

Egalité	**=**
Différent	**<>**
Inférieur strict	**<**
Inférieur ou égale	**<=**
Supérieur strict	**>**
Supérieur ou égale	**>=**

Ayant un résultat booléen

6.3. Les opérateurs logiques

Not : le "non"
And : le "et" logique des maths
Or : le "ou" logique
Xor : le "ou" exclusif

Pour les booléens.
*Aussi ; **Nand, Nor***

Tableau de vérité

A	B	Not A	Not B	A And B	A Or B	A Xor B	A Nand B	A Nor B
F	F	V	V	F	F	F	V	V
F	V	V	F	F	V	V	V	F
V	F	F	V	F	V	V	V	F
V	V	F	F	V	V	F	F	F

6.4. Les priorités dans les opérations

*En mathématiques une question se pose avec un calcul tel que **2 + 3 * 4** : quel est le résultat de ce calcul ?*
*Est-ce **20** (calcul **2 + 3 = 5** puis multiplication par **4**) ou **14** (ajouter **2** à **3 * 4**) ?*
Le même problème se pose en programmation. Il en existe deux solutions :

- Dans le cas d'une expression comportant des parenthèses, les parenthèses sont naturellement considérées en premier.
- En l'absence de parenthèses, des règles de priorités interviennent. Celles propres à **Pascal** sont :
  a. L'évaluation des fonctions est l'opération la plus prioritaire.
  b. Les opérateurs dits multiplicatifs (*** / Div Mod …**) sont plus prioritaires que les opérateurs dits additifs (**+ - …**).
  c. En cas de même priorité, l'expression est évaluée de gauche à droite.

Niveaux de priorité des opérateurs :

Niveau 1 : **Not**.
Niveau 2 : *****, **/**, **Div**, **Mod**, **And**.
Niveau 3 : **+**, **-**, **Or**, **Xor**.
Niveau 4 : **=**, **<**, **>**, **<=**, **>=**, **<>**.

Exemples :

`2+3*4`	donne **14** car la sous-expression **3*4** est d'abord évaluée (règle **b**: priorité de ***** par rapport à **+**)
`(2+3)*4`	**2+3** donne **5**, multiplié par **4** donne **20**
`2+3+4*2*5`	donne **45** (**4 * 2** puis multiplier par **5** donne **40**, puis **2** + **3** donne **5**, ensuite ajouter **40** à **5**)
`2+(3+4*2)*5`	donne **57** (**4 * 2** puis ajouter **3** donne **11**, multiplier par **5** puis ajouter **2**)
`4*Sin(π/2)*5.0`	calcul de **Sin (π/2)** puis multiplication par **4**, ensuite par **5.0** (règle **a**) (ça donne **20.0**)
`2*3+4 Div 3`	donne **7** car la sous-expression **2 * 3** est d'abord évaluée (règles **b** puis **c**), puis la sous-expression **4 Div 3** l'est (règles **b** puis **c**) enfin l'addition des résultats partiels **6** et **1** est effectuée
`(2*3+4)Div 3`	donne **3** car la sous-expression **2 * 3** est d'abord évaluée (règles **b**), puis ajouter **4** donne **10** (règles **b**), enfin **10 Div 3** est effectuée
`2*(3+4)Div 3`	donne **4** car la sous-expression **3 + 4** est d'abord évaluée donne **7**, puis multiplier **7 * 2** l'est (règles **b** puis **c**) donne **14** enfin **14 Div 3** est effectuée
`2*(3.0+4)Div 3`	donne **ERREUR** car la sous-expression **3.0 + 4** est d'abord évaluée donne **7.0**, puis multiplier **7.0 * 2** l'est (règles **b** puis **c**) donne **14.0** on ne peut pas effectuer l'opération **14.0 Div 3** (**14.0** est un réel)
`20/2*3`	**30.0** (Réel)

6.5. L'opérateur d'affectation

Une affectation consiste à attribuer une valeur à une variable.

La syntaxe générale est la suivante :

En **algorithmique** :

```
Nom_Variable ⇦ <Expression>
```

En **Pascal** :

```
<Identificateur> := <Expression> ;
```

<Expression> peut être :

- Une valeur constante — *exp* : **`surface := 40 ;`**
- Une autre variable — *exp* : **`Donnee := ValeurMemorisee ;`**
- Le résultat d'une fonction — *exp* : **`Resultat := Sqrt(Nombre) ;`**
- Un calcul portant sur ces différents éléments — *exp* : **`Surface := (Pi*Sqr (D))/4 ;`**

Exemples :

```
Algorithme Affectation
Variable A, B : Entier
Début
A ⇦ 1
B ⇦ A + 3
A ⇦ 3
Fin
```

A	B
1	?
1	4
3	4

Trace de l'Algorithme

Trace *: l'exécuter instruction par instruction (le faire tourner à la main) L'ordre des instructions est primordial.*

```
Début
A ⇦ "Salam"
B ⇦ "Merci"
Fin
```

Chaîne de caractère. ***B*** *contient « M e r c i »*

```
Début
A ⇦ "Salam"
B ⇦ Merci
Fin
```

Autre variable. ***B*** *contient le contenu de la variable* ***Merci****.*

7. Les opérations d'entrée/sortie

Permettent à la machine de dialoguer avec l'utilisateur.

7.1. Sortie (Ecriture, Affichage)

C'est l'instruction d'affichage et s'utilise comme suit :

En **algorithmique** :

```
     Ecrire (Nom_Variable)
Ou   Ecrire ("<Message>")
```

Ecrire *s'appelle aussi* ***Afficher****.*

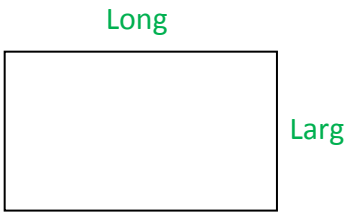
En **Pascal** :

```
     Write (<Identificateur>) ;
Ou   Write (<Valeur>) ;
Ou   Write ('<Message>') ;
Ou   Write ('<Message>',<Identificateur>) ;
```

Exemples :

```
Write('Bienvenue à tous') ;
Write ( 10 ) ;      {écrit la valeur 10 à l'écran}
Write (10, 4) ; {écrit la valeur 10 suivie de la valeur 4 sur l'écran}
Write ( 10 + 4 ) ; {écrit la valeur de l'expression, à savoir 14, sur l'écran}
Delta := 7 ;
Write('la valeur de Δ est',delta) ;
Write ('L''age de l''étudiant est 20 ans.') ;
Write(delta) ;
Write('delta') ;
```

Exemple : Ecrire un programme (Pascal et Algorithme) pour le calcul du périmètre d'un rectangle de longueur 12 cm et de largeur 7 cm.



En **algorithmique** :	En **Pascal** :
Algorithme Rectangle **Variable Long, Larg, Peri : Réel** **Début** **Long ⇦ 12** **Larg ⇦ 7** **Peri ⇦ (Long + Larg) * 2** **Ecrire (Peri)** **Fin**	**Program Rectangle ;** **Var Long, Larg, Peri : Real ;** **Begin** **Long := 12 ;** **Larg := 7 ;** **Peri := (Long + Larg) * 2 ;** **Write (Peri) ;** **End.**

Pour améliorer la présentation, nous fournissons des textes écrits entre guillemets " " (en Algo) ou entre apostrophes ' ' (en Pascal) ; comme :

```
Ecrire ("Le périmètre du rectangle est ", Peri, "cm")
```

Son exécution produit la ligne suivante :

```
Le périmètre du rectangle est 38 cm
```

- *Si le texte à afficher contient une apostrophe, il faut alors la doubler (en Pascal).*
- *Les différents noms de variables doivent êtres séparés par des virgules.*
- *L'identificateur est remplacé par la valeur de la variable correspondante au moment de l'exécution de l'instruction.*
- *La valeur peut être une constante ou une expression.*

Lorsque l'on veut aller à la ligne après un affichage, on utilise **Writeln** à la place de **Write**.

Exemple :

```
WriteLn ('Texte avec renvoi à la ligne') ;
```

***Remarque** : Les arguments d'une instruction **WriteLn** sont des expressions.*

7.2. Entrée (Lecture, Saisie)

*L'algorithme précédent pourrait être adapté à d'autres rectangles (d'autres **Long** et **Larg**) il est plus pratique d'utiliser l'action **Lire** qui permet de choisir la valeur à affecter à une variable au moment de l'exécution de l'algorithme.*

C'est l'instruction de saisie et s'utilise comme suit :

En **algorithmique** :

```
Lire (Nom_Variable)
```

En **Pascal** :

```
Read (<Identificateur>) ;
Ou   Read (V1, V2, V3, …) ;
```

Dès que le programme rencontre une instruction **Read**, l'exécution s'interrompt, attendant la frappe d'une valeur au clavier.

Remplaçons, dans notre programme, les 2 premières lignes par :

```
Read (Long) ;
Read (Larg) ;
```

Pour le confort de l'utilisateur, il est conseillé de précéder chaque instruction **Read** d'une instruction **Write** qui le prévient que le programme attend une valeur, et lui précise laquelle. Exemple :

```
Write ("Veuillez fournir la longueur du rectangle en cm")
Read (Long) ;
```

Il est conseillé d'utiliser **Readln** à la place de **Read** afin de supprimer le retour chariot (récupéré suite à la validation de l'utilisateur par la touche "Entrée") du tampon.

NB :

- **Readln(...) ;** ➔ passage à la ligne suivante en ignorant ce qui reste sur la ligne.
- **Readln ;** peut être employé sans paramètre.

Exemples :

```
Readln(a) ; Readln(b,c) ; Readln
```

*L'équivalent de la commande **ReadLn** est **ReadKey** qui donne une valeur à une variable de type « Char » (caractère **ASCII**).*

Syntaxe :

```
x := ReadKey ;
```

*Il existe une équivalence à cette commande très utile pour sortir d'une boucle : **KeyPressed**.*

Syntaxe :

```
Repeat
...
commandes
...
Until KeyPressed ;
```

Exemple : Ecrire un programme pour le calcul de la surface et du périmètre d'un cercle. (Pour la maison)

```
Program Surface_Perimetre_Cercle ;
Const Pi=3.14159 ;
Var Rayon, Surface, Peri : Real ;
Begin
Write (‘Entrer le rayon du cercle R = ‘) ;
Read (Rayon) ;
Surface := Pi * Sqr (Rayon) ; {Ou Pi * Rayon * Rayon}
Peri := Pi * 2 * Rayon ;
Writeln (‘La surface du cercle est ‘, Surface) ;
Writeln (‘Le périmètre du cercle est ‘, Peri) ;
End.
```