

Chapitre II : Notion d'algorithme et de programme

II.1. Objectif de ce chapitre

A l'issu de ce chapitre, l'apprenant sera capable de :

- Comprendre la définition et le rôle des algorithmes dans la résolution de problèmes informatiques.
- Identifier les différentes étapes d'un algorithme, de la spécification à l'exécution.
- Appréhender les concepts de variables, de types de données et d'opérations élémentaires dans la programmation.
- Analyser des exemples concrets d'algorithmes simples et comprendre comment ils sont traduits en programmes exécutables.

II.2. Concept d'un algorithme

Un Algorithme est une **séquence d'instructions ordonnées**, qui permet de **résoudre un problème**. Le terme “*Algorithme*” vient de l'arabe **الخوارزمي**, nom du mathématicien perse Al-Khwarizmi.

Un algorithme prend, en entrée, un ensemble de données (Inputs) et délivre (produit, renvoie) un ensemble de données en sortie (Outputs), afin de résoudre un problème.

Un algorithme peut être schématisé comme suit :

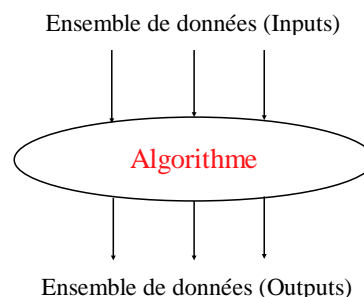


Figure II.1 : Structure d'un algorithme

Donc, un algorithme représente une solution pour un problème donné. Cette solution est spécifiée à travers un ensemble d'instructions (séquentielles avec un ordre logique) qui manipulent des données. Une fois l'algorithme est écrit (avec n'importe quelle langues : français, anglais, arabe, etc.), il sera transformé, après avoir choisi un langage de programmation, en un programme code source qui sera compilé (traduit) et exécuté par l'ordinateur.

Pour le langage de programmation qui sera utilisé, ça sera le langage **PASCAL**.

II.3. Les étapes de résolution d'un problème

La résolution d'un problème, en informatique, passe par différentes étapes. Ces dernières sont schématisées dans la figure II.2.

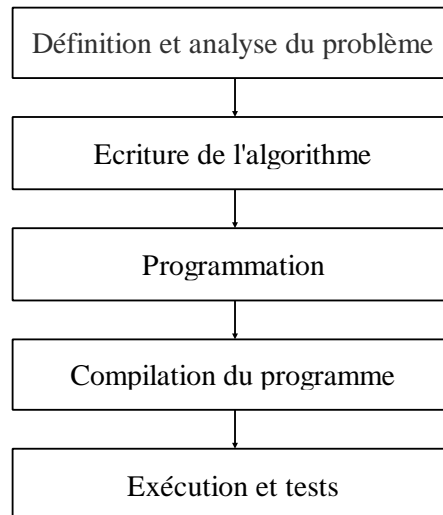


Figure II.2 : Etapes de résolution d'un problème

- **Définition et analyse du problème :** L'objectif de cette étape est de bien comprendre l'énoncé du problème et de déterminer :
 - Les résultats attendus (sorties),
 - Le traitement à effectuer (méthodes et formules de calculs pour atteindre le résultat),
 - Les données nécessaires au traitement (entrées).
- **Écriture de l'algorithme :** Une fois l'analyse terminée, il faut écrire les instructions dans leur ordre logique d'exécution et obtenir ainsi un algorithme. Ce dernier peut être écrit dans n'importe quelle langue.
- **Programmation :** C'est la traduction de l'algorithme en programme en utilisant un langage de programmation.
- **Langage de programmation :** C'est l'ensemble de la syntaxe (commandes et instructions) utilisé par les développeurs afin d'écrire un **programme**. Il existe plusieurs langages de programmation, comme C/C++, Pascal, Java, Fortran, Matlab, etc.
- **Compilation du programme :** Désigne le procédé de traduction d'un programme, écrit et lisible par un humain (code source), en un programme exécutable par un ordinateur (code binaire).

- **Exécution et tests** : Une fois compilé, un programme doit être testé pour s'assurer de son fonctionnement et qu'il répond aux besoins exprimés par l'utilisateur. Un programme est testé par des valeurs différentes de données (jeu de test).

II.4. Structure d'un algorithme

Un algorithme se compose de trois parties principales : L'entête, la partie déclarative et le corps de l'algorithme.

- **L'Entête** : Dans cette partie on déclare le nom de l'algorithme à travers un identificateur.
- **La partie déclarative** : Dans cette partie on déclare toutes les données utilisées par l'algorithme.
- **Le corps de programme** : Représente la séquence d'actions (instructions) Pour écrire un algorithme.

Ces parties doivent respecter une syntaxe bien déterminée, définie comme suit :

Tableau II.1 : Structure d'un algorithme

Algorithme	
L'Entête	{ Algorithme <identificateur_algo> ;
La partie déclarative	{ Constantes < identificateur_constant > = valeur ; Variables < identificateur_variable > : Type;
Le corps de programme	{ Début <Instruction 1> ; <Instruction 2> ; .. <Instruction n> ; Fin.

Tableau II.2 : Structure d'un algorithme en Pascal

Pascal	
L'Entête	{ Program <identificateur_algo> ;
La partie déclarative	{ Const < identificateur_constant > = valeur ; Var < identificateur_variable > : Type;
Le corps de programme	{ Begin <Instruction 1> ; <Instruction 2> ; .. <Instruction n> ; End.

II.4.1. L'Entête

L'entête sert à donner un nom à l'algorithme en utilisant un identificateur. Ce dernier est précédé par le mot clé "**Algorithme**". Alors qu'est-ce qu'un identificateur ?

- **Identificateur** : Un identificateur est une chaîne de caractère qui permet de donner un nom unique à un programme (algorithme), une constante, une variable, une procédure ou une fonction. Cette chaîne doit commencer soit par un caractère alphabétique ou par un tiret du 8 (_) et ne peut contenir que des caractères alphanumériques. Aussi, les mots réservés (mots-clés) d'un langage de programmation ne peuvent être utilisés comme identificateurs. Voici quelques mots réservés au langage Pascal: *Begin, end, program, var, const, real, integer, char, if, then, else, while, for, do, repeat*.

Exemples :

Tableau II.3 : Exemples d'identificateurs valides et non valides

Identificateur valide	Identificateur non valide
a1	x1 y (à cause du blanc ou l'espace)
a_1	x1-y (à cause du signe)
A_1	1xy (commence par un caractère numérique)
x12y	
x1_y	

II.4.2. La partie déclarative

La partie déclarative sert à déclarer les différentes données que l'algorithme utilise (Constantes, variables,.. etc.). Une donnée non déclarée et utilisée par l'algorithme engendre une erreur lors de la compilation. Alors qu'est-ce qu'une variable et qu'est-ce qu'une constante ?

- **Constantes** : Une constante est un objet contenant une valeur qui ne peut jamais être modifiée. Son objectif est d'éviter d'utiliser une valeur d'une manière direct. Imaginons qu'un algorithme utilise la valeur 3.14 une dizaines de fois (le nombre d'occurrences de la valeur 3.14 est par exemple 15) et qu'on veut modifier cette valeur par une autre valeur plus précise : 3.14159. Dans ce cas on est amené à modifier toutes les occurrences de 3.14. Par contre, si on utilise une constante $PI = 3.14$ on modifier une seule fois cette constante.

Exemples :

Ci-dessous quelques exemples de déclarations de constantes :

Const PI = 3.14; Constante réelle.
Const MAX = 10; Constante entière.
Const cc = 'a'; Constante caractère.
Const ss = 'algo'; Constante chaîne de caractère.
Const b1 = true; Constante booléenne.
Const b2 = false; Constante booléenne.

- **Variables** : Une variable est un objet contenant une valeur pouvant être modifiée.

Le tableau II.4 résume les 5 types de variables.

Tableau II.4 : Les 5 types de base

Algorithme	Pascal	Valeurs
Entier	Integer	Représente l'ensemble {..., 4, 3, 2, 1, 0, 1, 2, 3, 4, ...}
Réel	Real	Représente les valeurs numériques fractionnels et avec des virgule fixes (ou flottante)
Booléen	Boolean	Représente les deux valeurs <i>TRUE</i> et <i>FALSE</i> .
Caractère	Char	Représente tous les caractères imprimable.
Chaîne de caractères	String	Une séquence d'un ou plusieurs caractères

Exemples :

Tableau II.5 : Exemples de variables

Algorithme	Pascal	Signification
x : réel	x : real;	variable réelle
n, m : entier	n, m : integer;	deux variables entières
s : chaîne de caractères	s : String;	variables chaîne de caractères
b1, b2, b3 : boolean	b1, b2, b3 : boolean;	3 variables booléennes
c1 : caractère	c1 : char;	variable caractère

❖ **Remarques**

- Pour commenter un programme Pascal, on écrit les commentaires entre les accolades { }. Par exemple : {Ceci est un commentaire}.
- Dans un programme Pascal, on déclare les constantes dans une section qui commence par le mot clé **const**.
- Dans un programme Pascal, on déclare les variables dans une section qui commence par le mot

clé **var**.

- En plus des constantes et des variables, il est possible de déclarer de nouveaux types, des étiquettes, et (dans un programme Pascal) des fonctions et des procédures.

II.4.3. Le corps de programme

Le corps d'un algorithme est constitué d'un ensemble d'actions / instructions ordonnées de manière séquentielle et logique. Ces instructions se divisent en cinq types distincts:

- **Lecture** : Cette opération consiste à introduire des données dans l'algorithme. Une lecture consiste à donner une valeur arbitraire à une variable.
- **Écriture** : Cette opération implique l'affichage de données. Elle permet d'afficher des résultats ou des messages.
- **Affectation** : Elle permet de modifier les valeurs des variables en leur assignant de nouvelles valeurs.
- **Structures de contrôle** : Ces structures permettent de modifier la séquentialité de l'algorithme. Elles sont utilisées pour sélectionner différents chemins d'exécution ou pour répéter un traitement.
 - Structure de **Test alternatif simple / double**
 - Structure répétitives (itérative)
 - ✚ La boucle **Pour**
 - ✚ La boucle **Tant-que**
 - ✚ La boucle **Répéter**

Dans le langage Pascal, chaque instruction se termine par un **point-virgule**. Sauf à la fin du programme, on met un **point**.

II.5. Types d'instructions

Une instruction spécifie une opération ou un enchaînement d'opérations à exécuter sur des objets (constante, variable, .. etc.).

Les instructions se situent entre les mots-clés Début (Begin) et Fin (End.).

Les instructions sont séparées par des ; et sont exécutées séquentiellement, c'est-à-dire l'une après l'autre, depuis le Begin jusqu'au End. final.

II.5.1. Instructions d'Entrées/Sorties (Lecture / Écriture)

II.5.1.1. Entrées (Lecture)

Une instruction d'entrée est une instruction qui permet d'introduire (saisir) une valeur à

l'Algorithme à travers le clavier.

La syntaxe d'une lecture est comme suit :

Tableau II.6 : La syntaxe d'une lecture

Algorithme	Pascal
Lire (id_var) ;	read (id_var) ; Readln (id_var) ; {Lecture avec retour à la ligne.}
Lire (id_var1, id_var2, ..., id_varN) ; {Lecture de plusieurs variables en même temps.}	Read (id_var1, id_var2, id_var3, ..., id_varN) ;

❖ **Remarque :** Il est important de noter que l'instruction de lecture concerne uniquement les variables, on peut pas lire des constantes ou des valeurs. Lors de la lecture d'une variable dans un programme Pascal, le programme se bloque en attendant la saisie d'une valeur via le clavier. Une fois la valeur saisie, on valide par la touche *entrée*, et le programme reprend l'exécution avec l'instruction suivante.

Exemples :

Tableau II.7 : Exemples d'entrées

Algorithme	Pascal
Lire (a, b, c) lire (hauteur)	read(a, b, c); read(hauteur);

II.5.1.2. Sorties (Écriture)

L'écriture est une instruction qui permet d'afficher, à l'écran, des données. Ces dernières peuvent être un message, une valeur, la valeur d'une variable, une constante et même le résultat d'un calcul.

La syntaxe d'une écriture est comme suit :

Tableau II.8 : La syntaxe d'une écriture

Algorithme	Pascal
Écrire (id_var) ;	Write (id_var) ; Writeln (id_var) ; {Écriture avec retour à la ligne.}
Écrire (' Ceci est un message ! ') ; {Affichage du message : Ceci est un message ! à l'écran.}	Write (' Ceci est un message ! ') ;

❖ **Remarque :** Il est à noter que l'instruction d'écriture ne concerne pas uniquement les variables,

on peut écrire des constantes, valeurs ou des expressions (arithmétiques ou logiques). On peut afficher une valeur et sauter la ligne juste après à travers l'instruction : **writeln**.

Exemples :

Tableau II.9 : Exemples de sorties

Algorithme	Pascal	Signification
écrire('Bonjour')	write ('Bonjour');	Afficher le message Bonjour
écrire(a, b, c)	write(a, b, c);	Afficher les valeurs des variables a, b et c
écrire(5+2)	write(5+2);	Afficher le résultat du calcul c-à-d 7
écrire(a+b-c)	write(a+b-c);	Afficher le résultat de l'évaluation de l'expression : a+b-c
écrire(5<2)	write(5<2);	Afficher le résultat de l'évaluation de l'expression 5 < 2 c-à-d false.

❖ **Remarque :** En Pascal, il est possible de commenter un programme, il suffit d'écrire les commentaires entre accolades { } ou de mettre deux slashes // avant le commentaire.

Par exemple : { Ceci est un commentaire } ou // Ceci est un commentaire.

Le commentaire n'est pas pris en compte à la compilation. Il sert à rendre le programme plus clair à la lecture, à noter des remarques, etc

II.5.2. Instruction d'affectation

L'affectation est une instruction qui permet de modifier la valeur d'une variable.

La syntaxe d'une affectation est :

Tableau II.10 : La syntaxe d'une affectation

Algorithme	Pascal
$a \leftarrow b$; {a : DOIT être une variable. b peut être une valeur, une constante, une variable ou une expression.}	$a := b$;

❖ **Remarque :** Les deux côtés d'une affectation doivent être du même type sauf pour le type entier qui peut être stocké dans un réel car l'ensemble des réels inclut l'ensemble des entiers.

Exemples :**Tableau II.11 :** Exemples d'affectation

Algorithme	Pascal	Signification
$a \leftarrow 5$	$a:=5;$	Mettre la valeur 5 dans la variable a
$b \leftarrow a+5$	$b:=a+5;$	Mettre la valeur de l'expression a+5 dans la variable B
$sup \leftarrow a>b$	$sup:=a>b;$	a>b donne un résultat booléen, donc sup est une variable booléenne

II.5.3. Structures de contrôles

En générale, les instructions d'un programme sont exécutés d'une manière séquentielle : la première instruction, ensuite la deuxième, après la troisième et ainsi de suite. Cependant, dans plusieurs cas, on est amené soit à choisir entre deux ou plusieurs chemins d'exécution (un choix entre deux ou plusieurs options), ou bien à répéter l'exécution d'un ensemble d'instructions, pour cela nous avons besoins de structures de contrôle pour contrôler et choisir les chemins d'exécution ou refaire un traitement plusieurs fois. Les structures de contrôle sont de deux types : Structures de contrôles conditionnelles et structures de contrôle répétitives (itératives).

II.5.3.1. Structures de contrôle conditionnelle

Ces structures sont utilisées pour décider ou pas de l'exécution d'un ou de plusieurs instructions en testant (vérifiant) une ou plusieurs condition. On en distingue deux types de tests : le test alternatif simple et double.

a) Test alternatif simple

Ce test contient un seul bloc d'instructions. Selon une condition (expression logique), on décide est ce que le bloc d'instructions est exécuté ou non. Si la condition est vraie, on exécute le bloc, sinon on l'exécute pas.

La syntaxe d'un test alternatif simple est comme suit :

Tableau II.12 : La syntaxe d'un test alternatif simple

Algorithme	Pascal
si <condition> alors <instruction(s)> finsi;	if <condition> then begin <instruction(s)>; end;

Exemples :**Tableau II.13 :** Exemple d'un test alternatif simple

Algorithme	Pascal
<pre> lire(x) si x > 2 alors x ← x + 3 ; finsi ; écrire (x) </pre>	<pre> read(x); if x > 2 then begin x:= x + 3; end; write(x); </pre>

❖ **Remarque :** En Pascal, le bloc d'instructions à exécuter après if (juste après then) **DOIT** être délimité par un **begin** et **end**.

Si le bloc contient une seule instruction, **begin** et **end** sont facultatifs (on peut les enlever).

b) Test alternatif double

Ce test contient deux blocs d'instructions. Le premier bloc est exécuté lorsque la condition est vérifiée (vraie) et le second lorsque la condition n'est pas vérifiée (fausse).

La syntaxe d'un test alternatif double est :

Tableau II.14 : La syntaxe d'un test alternatif double

Algorithme	Pascal
<pre> si <condition> alors <instruction(s)1> sinon <instrucion(s)2> ; finsi ; </pre>	<pre> if <condition> then begin <instruction(s)1> ; end else begin <instruction(s)2> ; end ; </pre>

Exemple :**Tableau II.15 :** Exemples d'un test alternatif double

Algorithme	Pascal
<pre> lire(x) si x > 2 alors x ← x + 3 sinon x ← x - 2 finsi écrire (x) </pre>	<pre> read(x); if x > 2 then begin x:= x + 3; end else begin x:= x - 2; end; write(x); </pre>

❖ **Remarques :**

- En Pascal, l'instruction qui précède **else** ne doit pas contenir un point-virgule (;).
- Dans l'exemple précédent, on peut enlever **begin** et **end** du **if** et ceux du **else** puisqu'il y a une seule instruction dans les deux blocs.

II.5.3.2. Structures de contrôle répétitives

Les structures répétitives nous permettent de répéter un traitement un nombre fini de fois. Par exemple, on veut afficher tous les nombres premiers entre 1 et N (N nombre entier positif donné).

Nous avons trois types de structures itératives (boucles) :

a) Boucle Pour (For)

La structure de contrôle répétitive pour (for en langage Pascal) utilise un indice entier qui varie (avec un incrément = 1) d'une valeur initiale jusqu'à une valeur finale. À la fin de chaque itération, l'indice est incrémenté de 1 d'une manière automatique (implicite).

La syntaxe de la boucle **pour** est comme suit :

Tableau II.16 : La syntaxe de la boucle **pour**

Algorithme	Pascal
pour <cpt> ← <vi> à <vf> faire <instruction(s)> ; finPour ;	for <cpt> := <vi> to <vf> do begin <instruction(s)> ; end ;

<cpt> : le compteur (variable entière)

<vi> : valeur initiale <vf> : valeur finale

La boucle pour contient un bloc d'instructions (les instructions à répéter). Si le bloc contient une seule instruction, le **begin** et **end** sont facultatifs.

Le bloc sera répété un nombre de fois = (<vf> - <vi> + 1) si la valeur finale est supérieure ou égale à la valeur initiale. Le bloc sera exécuté pour <cpt> = <vi>, pour <cpt> = <vi>+1, pour <cpt> = <vi>+2, ..., pour <cpt> = <vf>.

❖ **Remarque :** Il ne faut jamais mettre de point-virgule après le mot clé **do**. (erreur logique)

b) Boucle Tant-que (While)

La structure de contrôle répétitive **tantque** (**while** en langage Pascal) utilise une expression

logique ou booléenne comme condition d'accès à la boucle : si la condition est vérifiée (elle donne un résultat vrai : TRUE) donc on entre à la boucle, sinon on la quitte.

La syntaxe de la boucle **tantque** est comme suit :

Tableau II.17 : La syntaxe de la boucle **tantque**

Algorithme	Pascal
tant-que <condition> faire <instruction(s)> ; fin tant-que ;	while <condition> do begin <instruction(s)>; end ;

<condition> : expression logique qui peut être vraie ou fausse.

On exécute le bloc d'instructions Tant que la condition est vraie. Une fois la condition est fausse, on arrête la boucle, et on continue l'exécution de l'instruction qui vient après fin Tant que (après **end**).

Comme la boucle **for**, il faut jamais mettre de point-virgule après **do**.

❖ **Remarque :** Il est possible de remplacer toute boucle "**pour**" par une boucle "**tantque**", cependant, l'inverse n'est pas toujours réalisable.

c) Boucle Répéter (Repeat)

La structure de contrôle répétitive **répéter** (**repeat** en langage Pascal) utilise une expression logique ou booléenne comme condition de sortie de la boucle : si la condition est vérifiée (elle donne un résultat vrai : TRUE) on sort de la boucle, sinon on y accède (on répète l'exécution du bloc).

La syntaxe de la boucle **répéter** est comme suit :

Tableau II.18 : La syntaxe de la boucle **répéter**

Algorithme	Pascal
répéter <instruction(s)> ; Jusqu'à <condition> ;	repeat <instruction(s)>; until <condition> ;

<condition> : expression logique qui peut être vraie ou fausse.

On exécute le bloc d'instructions jusqu'à avoir la condition correcte. Une fois la condition est vérifiée, on arrête la boucle, et on continue l'exécution de l'instruction qui vient après **jusqu'à** (après **until**). Dans la boucle **repeat** on utilise pas **begin** et **end** pour délimiter le bloc d'instructions (le bloc est déjà délimité par **repeat** et **until**).

La différence entre la boucle **répéter** et la boucle **tantque** est :

- La condition de **répéter** est toujours l'inverse de la condition **tantque** : pour **répéter** c'est la condition de sortie de la boucle, et pour **tantque** c'est la condition d'entrer.
- Le teste de la condition est à la fin de la boucle (la fin de l'itération) pour **répéter**. Par contre, il est au début de l'itération pour la boucle **tantque**. C'est-à-dire, dans **tantque** on teste la condition avant d'entrer à l'itération, et dans **répéter** on fait l'itération après on teste la condition.

II.5.3.3. Structure de contrôle de branchements / sauts (l'instruction Goto)

Une instruction de branchement nous permet de sauter à un endroit du programme et continuer l'exécution à partir de cet endroit. Pour réaliser un branchement, il faut tout d'abord indiquer la cible du branchement via une étiquette `<num_etiq> : .`. Après on saute à cette endroit par l'instruction aller à `<num_etiq>` (en pascal : `goto <num_etiq>`).

La syntaxe d'un branchement est comme suit :

Tableau II.19 : La syntaxe d'un branchement

Algorithme	Pascal
aller à <code><num_etiq></code> . . . <code><num_etiq> : .</code> . .	goto <code><num_etiq>;</code> . . . <code><num_etiq> : .</code> . .

N.B :

- Une étiquette représente un numéro (nombre entier), exemple : 1, 2, 3, etc.
- Dans un programme Pascal, il faut déclarer les étiquettes dans la partie déclaration avec le mot clé **label**. (on a vu **const** pour les constantes **var** pour les variables)
- Une étiquette désigne un seule endroit dans le programme, on peut jamais indiquer deux endroits avec une même étiquette.
- Par contre, on peut réaliser plusieurs branchement vers une même étiquette.
- Un saut ou un branchement peut être vers une instruction antérieure ou postérieure (avant ou après le saut).

Exemple :**Tableau II.20 :** Exemple de branchement

Algorithme	Pascal
algorithme branchement variables a, b, c : entier ; début lire (a, b); 2: c \leftarrow a; si (a > b) alors aller à 1; finsi a \leftarrow a + 5; aller à 2; 1: écrire (c); fin	program branchement; uses wincrt; var a, b, c:integer; label 1, 2; begin read(a,b); 2: c:=a; if (a>b) then goto 1; a := a + 5; goto 2; 1: write(c); end.

Dans l'exemple ci-dessus, il y a deux étiquettes : **1** et **2**. L'étiquette **1** fait référence la dernière instruction de l'algorithme/programme (écrire(c) /write(c)), et l'étiquette **2** fait référence la troisième instruction de l'algorithme / programme (c \leftarrow a; / c := a;). Pour le déroulement de l'algorithme, on utilise le tableau suivant (a = 2 et b = 5) :

Tableau II.21 : Déroulement de l'algorithme

Variables / Instructions	a	b	c
Lire (a, b) Donner deux valeur quelconque à a et b	2	5	?
c \leftarrow a ;	2	5	2
a > b \rightarrow false puisque a = 2 et b =5 on entre pas aubloc du si a \leftarrow a + 5;	7	5	2
aller à 2 c \leftarrow a;	7	5	7
a > b \rightarrow true puisque a = 7 et b =5 on entre au blocdu si aller à 1 =>écrire (c)	7	5	7 (résultat affiché)

Il y a deux types de branchement :

- a. **branchement inconditionnel** : c'est un branchement sans condition, il n'appartient pas à un bloc de *si* ou un bloc *sinon*. Dans l'exemple précédent, l'instruction **aller à 2 (goto 2)** est un saut inconditionnel.
- b. **branchement conditionnel** : Par contre, un branchement conditionnel est un saut qui appartient à un bloc *si* ou un bloc *sinon*. L'instruction **aller à 1 (goto 1)**, dans l'exemple précédent est un saut conditionnel puisque il appartient un bloc *si*.

II.6. Correspondance Algorithme-Pascal

Pour traduire un algorithme en programme Pascal, on utilise le tableau récapitulatif suivant pour traduire chaque structure syntaxique d'un algorithme en structure syntaxique du Pascal.

Tableau II.22 : Correspondance Algorithme-Pascal

Vocabulaire / Syntaxe Algorithmique	Vocabulaire / Syntaxe du PASCAL
Algorithme	Program
Constantes	Const
Type	Type
Variables	Var
Etiquette	Label
Entier	Integer
Réel	Real
Caractère	Char
Booléen	Boolean
Chaîne de Caractères	String
Fonction	Function
Procédure	Procedure
Début	Begin
Fin	End
Si ... Alors ... Sinon ...	If ... Then ... Else ...
Tant-que ... Faire ...	While ... Do ...
Pour i ← 1 à N Faire ...	For i:= 1 To N Do ...
Pour i ← N à Pas- 1 Faire ...	For i:= 1 DownTo 1 Do ...
Répéter ... Jusqu'à ...	Repeat Until ...

❖ **Remarques importantes**

1. Langage Pascal est insensible à la casse, c'est-à-dire, si on écrit begin, Begin ou BEGIN c'est la même chose.
2. Lorsque l'action après THEN, ELSE ou un DO comporte plusieurs instructions, on doit obligatoirement encadrer ces instructions entre BEGIN et END. Autrement dit, on les définit sous forme d'un bloc. Pour une seule instruction, il n'est pas nécessaire (ou obligatoire) de l'encadrer entre BEGIN et END (voir en travaux pratiques). Un ensemble d'instructions encadrées entre BEGIN et END, s'appelle un BLOC ou action composée. On dit qu'un programme Pascal est structurée en blocs.
3. Il est interdit de chevaucher deux structures de boucles ou de blocs. Par exemple :

```

FOR..... DO
  BEGIN
    .....
    WHILE..... DO
      BEGIN
        .....
        .....
      END;
    END;
  END;

```

On a eu la forme suivante :

```

Boucle ou bloc 1
(la boucle For)  [
                  [
                    Boucle ou bloc 2
                    (la boucle While)
                  ]
                ]

```

Ce qui est interdit.

Les boucles et blocs doivent en aucun cas chevaucher, ils doivent être imbriqués.

Exemples de structures autorisées :

```

[ [ Exemple de deux blocs
[ [
[ [ Exemple de six blocs
[ [
[ [
[ [
[ [
[ [

```

II.7. Représentation en organigramme

Un organigramme est la représentation graphique de la résolution d'un problème. Il est similaire à un algorithme. Chaque type d'action dans l'algorithme possède une représentation dans l'organigramme.

Il est préférable d'utiliser la représentation algorithmique que la représentation par organigramme notamment lorsque le problème est complexe.

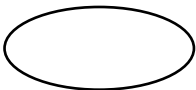
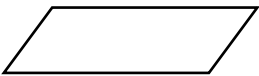
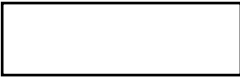
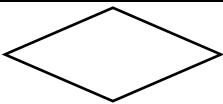


Les inconvénients qu'on peut rencontrer lors de l'utilisation des organigrammes sont :

- Quand l'organigramme est long et tient sur plus d'une page,
- Problème de chevauchement des flèches,
- Plus difficile à lire et à comprendre qu'un algorithme.

II.7. 1. Les symboles d'organigramme

Les symboles utilisés dans les organigrammes sont illustrés dans le tableau II.23.

Tableau II.23 : Les symboles d'organigramme

	Représente le début et la Fin de l'organigramme
	Entrées / Sorties : Lecture des données et écriture des résultats.
	Calculs, traitements
	Tests et décision : on écrit le test à l'intérieur du losange
	Ordre d'exécution des opérations (Enchaînement)
	Connecteur

II.8. Représentation des primitives algorithmiques

II.8.1. L'enchaînement

L'enchaînement permet d'exécuter une série d'actions dans l'ordre de leur apparition. Soit A_1, A_2, \dots, A_n une série d'actions, leur enchaînement est représenté comme suit :



$A1, A2, \dots, A_n$: peuvent être des actions élémentaires ou complexes.

II.8.2. La structure alternative simple

La syntaxe et l'organigramme de la structure alternative simple sont présentés dans le tableau II.24.

Tableau II.24 : La syntaxe et l'organigramme de la structure alternative simple

Représentation algorithmique	Représentation sous forme d'organigramme
si <condition> alors <Action(s)>; finsi ; Si la condition est vérifiée, le bloc <action(s)> sera exécuté, sinon rien, et on continue l'exécution de l'instruction après fin si.	<pre> graph TD Entry(()) --> Condition{Condition ?} Condition -- Vrai --> Action[Action(s)] Condition -- Faux --> Exit(()) Action --> Exit </pre>

Les conditions utilisées pour les tests (simple ou double) sont des expressions logiques ou booléennes, ça veut dire des expressions dont leur évaluation donne soit TRUE (Vrai) ou FALSE (faux). Toute comparaison entre deux nombres représente une expression logique. On peut former des expressions logiques à partir d'autres expressions logiques en utilisant les opérateurs suivants : Not, Or et And.

Exemple :

$(x \geq 5)$: est une expression logique, elle est vraie si la valeur de x est supérieure ou égale à 5. elle est fautive dans le cas contraire.

$\text{Not}(x \geq 5)$: est une expression logique qui est vraie uniquement si la valeur de x est inférieure à 5.

$(x \geq 5) \text{ And } (y \leq 0)$: est une expression logique qui est vraie si x est supérieure ou égale à 5 et y inférieure ou égale à 0.

II.8.3. La structure alternative double

La syntaxe et l'organigramme de la structure alternative double sont présentés dans le tableau II.25.

Tableau II.25 : La syntaxe et l'organigramme de la structure alternative double

Représentation algorithmique	Représentation sous forme d'organigramme
si <condition> alors <Action1(s)> sinon <Action2(s)>; finsi ; Si la condition est vérifiée, le bloc <action1(s)> sera exécuté, sinon (si elle est fausse) on exécute <action2(s)>.	<pre> graph TD Cond{Condition ?} -- Faux --> Act2[Action2(s)] Cond -- Vrai --> Act1[Action1(s)] Act2 --> Join(()) Act1 --> Join Join --> Exit(()) </pre>

II.8.4. La structure itérative POUR (Boucle POUR)

La syntaxe et l'organigramme de la structure itérative POUR sont présentés dans le tableau II.26.

Tableau II.26 : La syntaxe et l'organigramme de la structure itérative POUR

Représentation algorithmique	Représentation sous forme d'organigramme
pour <cpt> ← <vi> à <vf> faire <Action(s)>; finpour ;	<pre> graph TD Init[<cpt> ← <Vi>] --> Cond{<cpt> ≤ <Vf>} Cond -- Vrai --> Act[Action(s)] Act --> Inc[<cpt> ← <cpt> + 1] Inc --> Cond Cond -- Faux --> Exit(()) </pre>

Dans la boucle **POUR**, on exécute le bloc <Acitons> ($<vf> - <vi> + 1$) fois. Ceci dans le cas où $<vf>$ est supérieur ou égale à $<vi>$. Dans le cas contraire, le bloc d'actions ne sera jamais exécuté. Le déroulement de la boucle POUR est exprimé comme suit :

1. La variable entière <cpt> (le compteur) prends la valeur initiale <vi> ;
2. On compare la valeur de <cpt> à celle de <vf> ; si <cpt> est supérieur à <vf> on sort de la boucle ;
3. Si <cpt> est inférieur ou égale à <vf> on exécute le bloc <Action(s)> ;

4. La boucle POUR incrémente automatiquement le compteur $\langle cpt \rangle$, c'est-à-dire elle lui ajoute un ($\langle cpt \rangle \leftarrow \langle cpt \rangle + 1$);
5. On revient à 2 (pour refaire le teste $\langle cpt \rangle \leftarrow \langle vi \rangle$ C'est pour cela qu'on dit la boucle);

❖ **Remarque :** La boucle **POUR** est souvent utilisée pour les structures de données itératives (les tableaux et les matrices – variables indicées).

II.8.5. La structure itérative Tant-que (Boucle Tant-que)

La syntaxe et l'organigramme de la structure itérative Tant-que sont présentés dans le tableau II.27.

Tableau II.27 : La syntaxe et l'organigramme de la structure itérative Tant-que

Représentation algorithmique	Représentation sous forme d'organigramme
Tant-que $\langle \text{condition} \rangle$ faire $\langle \text{Action(s)} \rangle$; finpour ;	<pre> graph TD Entry(()) --> Cond{Condition ?} Cond -- Vrai --> Action[Action(s)] Action --> Cond Cond -- Faux --> Exit(()) </pre>

On exécute le bloc d'instructions $\langle \text{Action(s)} \rangle$ tant que la $\langle \text{condition} \rangle$ est vérifiée (c'est-à-dire elle est vraie). Le déroulement de la boucle est comme suit :

1. On évalue la condition : si la condition est fausse on sort de la boucle ; Si la condition est vraie, on exécute le bloc $\langle \text{Action(s)} \rangle$;
2. On revient à 1 ;
3. On continue la suite de l'algorithme

II.8.6. La structure itérative Répéter (Boucle Répéter)

La syntaxe et l'organigramme de la structure itérative répéter sont présentés dans le tableau II.28.

Tableau II.28 : La syntaxe et l'organigramme de la structure itérative répéter

Représentation algorithmique	Représentation sous forme d'organigramme
Répéter $\langle \text{Action(s)} \rangle$; Jusqu'à $\langle \text{condition} \rangle$;	<pre> graph TD Entry(()) --> Action[Action(s)] Action --> Cond{Condition ?} Cond -- Faux --> Action Cond -- Vrai --> Exit(()) </pre>

On répète l'exécution du bloc $\langle \text{Action(s)} \rangle$ jusqu'à avoir la condition correcte. Le déroulement est comment suit :

1. On exécute le bloc $\langle \text{Action(s)} \rangle$;
2. On évalue la condition : si la condition est vérifiée (elle est vraie) on sort de la boucle (on continue la suite de l'algorithme);
3. Si la condition n'est pas vérifiée (elle est fausse) on revient à 1.

❖ **Remarques :**

- ✓ N'importe quelle boucle **POUR** peut être remplacée par une boucle **Tant-Que**, cependant l'inverse n'est pas toujours correcte, c'est-à-dire, il y a des cas où la boucle **Tant-Que** ne peut pas être remplacée par une boucle **POUR**.
- ✓ On transforme une boucle **POUR** à une boucle **Tant-Que** comme suit :

Tableau II.29 : Transformation de la boucle POUR à la boucle Tant-que

Boucle POUR	Boucle Tant-Que
<p>pour $\langle \text{cpt} \rangle \leftarrow \langle \text{vi} \rangle$ à $\langle \text{vf} \rangle$ faire</p> <p style="padding-left: 40px;">$\langle \text{action(s)} \rangle$;</p> <p>Finpour;</p>	<p>$\langle \text{cpt} \rangle \leftarrow \langle \text{vi} \rangle$;</p> <p>Tant-que $\langle \text{cpt} \rangle \leftarrow \langle \text{vf} \rangle$ faire</p> <p style="padding-left: 40px;">$\langle \text{Action(s)} \rangle$;</p> <p style="padding-left: 40px;">$\langle \text{cpt} \rangle \leftarrow \langle \text{cpt} \rangle + 1$;</p> <p>Fin Tant-Que;</p>

- ✓ La boucle Répéter possède une condition de sortie (c'est-à-dire si elle est vraie on sort de la boucle), alors que la boucle Tant-que possède une condition d'entrée (c'est-à-dire si elle est vraie on entre dans la boucle).
- ✓ La boucle Répéter exécute le bloc $\langle \text{Action(s)} \rangle$ au moins une fois, le teste vient après l'exécution du bloc.
- ✓ La boucle Tant-Que peut ne pas exécuter le bloc $\langle \text{Action(s)} \rangle$ (dans le cas où la condition est fausse dès le début), puisque le teste est avant l'exécution du bloc.

II.9. Exercices corrigés

Exercice N°01 : (Type de variables)

Donner le type des variables suivantes : 2010 ; 124.5 ; 667.0E-8 ; 'A' ; TRUE ; False ; 'division par zéro'

Corrigé de l'exercice N°01 :

Type des variables

Variable	Type
2010	Entier / Integer
124.5	Réel / Real
667.0E-8	Réel / Real
'A'	Caractère / Char
TRUE	Booléen / Boolean
False	Booléen / Boolean
'division par zéro'	Chaîne de caractère / String

Exercice N°02 : (Identificateurs)

Identifier les identificateurs valides et non valides : 1A ; R? ; K2 ; T280 ; 12R ; Hauteur ; Prix-HT ; Prix_HT ; Exo 04 ; Exo_04 ; Exo-04 ; Program ; read.

Corrigé de l'exercice N°02 :

Les variables valides et non valides :

Variable valide	Variable non valide
K2	1A
T280	R?
Hauteur	12R
Prix_HT	Prix-HT
Exo_04	Exo 04
	Exo-04
	Program
	read

Exercice N°03 : (Enoncé du problème → Algorithme → Programme)

Écrire un algorithme, puis traduit le en programme PASCAL, pour chacun des problèmes suivants :

- 1) Permuter entre les deux variables X et Y ?
- 2) Permuter entre les trois variables X, Y et Z de telle sorte que la valeur de X soit dans Y, celle de Y dans Z et la valeur de Z dans X ?

- 3) Calculer la division entre deux nombres réels a et b ? (**NB** : sans faire de condition, c-à-d : $b \neq 0$)
- 4) Calculer le quotient et le reste de la division euclidienne de a par b ?
- 5) Convertir en octets un nombre donné en bits ?

Corrigé de l'exercice N°03 :

- 1) Permuter entre les deux variables X et Y ?

Algorithme	Programme PASCAL
Algorithme Exo2_1; Variables x, y, t : entier; Début {--*-- Entrées --*--} Lire (x, y) ; {--*-- Traitement --*--} t ← x; x ← y; y ← t; {--*-- Sorties --*--} Écrire('x=', x, 'y=', y) ; Fin.	Program Exo2_1; Var x, y, t : integer; Begin {--*-- Entrées --*--} Read (x,y) ; {--*-- Traitement --*--} t := x; {on conserve la valeur de X dans t} x := y; {pas de risque de perte de valeur} y := t; {on récupère l'ancienne valeur de X} {--*-- Sorties --*--} Write('x=', x, 'y=', y); End.

- 2) Permuter entre les trois variables X, Y et Z de telle sorte que la valeur de X soit dans Y, celle de Y dans Z et la valeur de Z dans X ?

Algorithme	Programme PASCAL
Algorithme Exo2_2; Variables x, y, z, t : entier; Début {--*-- Entrées --*--} Lire (x, y, z) ; {--*-- Traitements --*--} t ← y; y ← x; x ← z; z ← t; {--*-- Sorties --*--} Écrire('x=', x, 'y=', y, 'z=', z); Fin.	Program Exo2_2; Var x, y, z, t : integer; Begin {--*-- Entrées --*--} Read (x, y, z) ; {--*-- Traitements --*--} t := y; {on conserve la valeur de y dans t} y := x; {x dans y} x := z; {z dans x} z := t; {y dans z} {--*-- Sorties --*--} Write('x=', x, 'y=', y, 'z=', z); End.

3) Calculer la division entre deux nombres réels a et b ?

Algorithme	Programme PASCAL
Algorithme Exo2_3; Variables a, b, c : réel; Début {--*-*- Entrées --*-*-} Lire (a, b) ; {--*-*- Traitements --*-*-} $c \leftarrow a / b$; {--*-*- Sorties --*-*-} Écrire (c) ; Fin.	Program Exo2_3; Var a, b, c : real; Begin {--*-*- Entrées --*-*-} Read (a, b) ; {--*-*- Traitements --*-*-} $c := a / b$; {--*-*- Sorties --*-*-} Write (c) ; End.

4) Calculer le quotient et le reste de la division euclidienne de a par b ?

Algorithme	Programme PASCAL
Algorithme Exo2_4; Variables a, b, Q, R : entier; Début {--*-*- Entrées --*-*-} Lire (a, b) ; {--*-*- Traitements --*-*-} $Q \leftarrow a \text{ div } b$; $R \leftarrow a \text{ mod } b$; {--*-*- Sorties --*-*-} Écrire ('Le quotient est : ', Q, 'et le reste est : ', R) ; Fin.	Program Exo2_4; Var a, b, Q, R : integer; Begin {--*-*- Entrées --*-*-} Read (a, b) ; {--*-*- Traitements --*-*-} $Q := a \text{ div } b$; $R := a \text{ mod } b$; {--*-*- Sorties --*-*-} Write ('Le quotient est : ', Q, 'et le reste est : ', R) ; End.

5) Convertir en octets un nombre donné en bits ?

Algorithme	Programme PASCAL
Algorithme Exo2_7; Variables bit : integer ; octet : réel ; Début {--*-- Entrées --*--} Écrire ('Nombres de bits =') ; Lire (bit) ; {--*-- Traitements --*--} octet ← bit/8; {--*-- Sorties --*--} Écrire (bit, ' bits =', octet:8:3,' octet'); Fin.	Program Exo2_7; Var bit : integer ; octet : real ; Begin {--*-- Entrées --*--} Write ('Nombres de bits =') ; Read (bit) ; {--*-- Traitements --*--} octet := bit/8; {--*-- Sorties --*--} Write (bit, ' bits =', octet:8:3,' octet'); End.

Exercice N°04 :

Ecrire un programme Pascal intitulé **ordre_croissant**, qui permet d'afficher trois valeurs numérique A, B et C avec ordre croissant ?

Corrigé de l'exercice N°04 :

Algorithme	Programme PASCAL
Algorithme ordre_croissant; Variables A, B, C : entier; Début { -*-*- Entrées -*-*- } Écrire('Donner trois valeurs entière A, B et C : '); Read(A, B, C) ; { -*-*- Traitement & Sorties -*-*- } Si (A <= B) ET (B <= C) alors Écrire(A, B, C); Sinon Si (A <= C) ET (C <= B) alors Écrire (A, C, B) ; Sinon Si (B <= A) ET (A <= C) alors Écrire(B, A, C) Sinon Si (B <= C) ET (C <= A) alors Écrire(B, C, A) Sinon Si (C <= A) ET (A <= B) alors Écrire(C, A, B); Sinon Écrire(C, B, A); Fin-Si ; Fin-Si ; Fin-Si ; Fin-Si ; Fin.	Program ordre_croissant; Var A, B, C : integer; Begin { -*-*- Entrées -*-*- } Write('Donner trois valeurs entière A, B et C : '); Read(A, B, C) ; { -*-*- Traitement & Sorties -*-*- } Writeln('Les 3 valeurs avec un ordre croissant : ') ; if (A <= B) And (B <= C) then Write(A, B:5, C:5) else if (A <= C) And (C <= B) then Write(A, C:5, B:5) else if (B <= A) And (A <= C) then Write(B, A:5, C:5) else if (B <= C) And (C <= A) then Write(B, C:5, A:5) else if (C <= A) And (A <= B) then Write(C, A:5, B:5) else Write(C, B:5, A:5); End.

NB : Il existe d'autres solutions qui permettent d'afficher trois valeurs numériques A, B et C avec ordre croissant.

Exercice N°05 :

Ecrire un programme Pascal intitulé **PARITE** qui saisit un nombre entier et détecte si ce nombre est pair ou impair.

Corrigé de l'exercice N°05 :

Algorithme	Programme PASCAL
Algorithme Partie ; Variables N : entier; Début {--*--*-- Entrées --*--*--} Écrire ('Donner un entier : '); Lire (N) ; {--*--*-- Traitement & Sorties--*--*--} Si N mod 2 = 0 alors Écrire (N, ' est pair') Sinon Écrire (N, ' est impair') ; Fin-Si ; Fin.	Program Parite ; Var N : Integer ; Begin {--*--*-- Entrées --*--*--} Writeln ('Donner un entier : '); Readln (N) ; {--*--*-- Traitement & Sorties--*--*--} IF N mod 2 = 0 Then Writeln (N, ' est pair') Else Writeln (N, ' est impair') ; End.

Exercice N°06 :

Ecrire un algorithme/programme Pascal pour chaque cas suivant :

- 1) Calculer la somme $S = 1^2 + 3^2 + 5^2 + \dots + (2N + 1)^2$
- 2) Calculer le produit $P = 1 * 2 * 3 * \dots * N$
- 3) Calculer la somme $S = X + X^2 + X^3 + \dots + X^N$
- 4) Calculer la somme $S = x + \frac{x^3}{2} + \frac{x^5}{4!} + \frac{x^7}{6!} + \dots + (N^{\text{ème}} \text{ terme})$

Corrigé de l'exercice N°06 :

- 1) Calculer la somme $S = 1^2 + 3^2 + 5^2 + \dots + (2 * N + 1)^2$

Algorithme	Programme PASCAL
Algorithme exo2_1; variables S, N, i: entier; Début	Program exo2_1; Var S, N, i: integer; Begin

<pre> {-*-*- Entrées -*-*-} Ecrire('Donner la valeur de N : '); Lire(N); {-*-*- Traitement -*-*-} S ← 0; Pour i ← 0 à N faire S ← S + sqr(2*i+1); Fin-pour {-*-*- Sortie -*-*-} Ecrire ('S =', S); Fin. </pre>	<pre> {-*-*- Entrées -*-*-} Writeln('Donner la valeur de N : '); Readln(N); {-*-*- Traitement -*-*-} S:=0; For i:=0 to N do S:= S + sqr(2*i+1); {-*-*- Sortie -*-*-} Write('S =', S); End. </pre>
--	---

2) Calculer le produit $P = 1 * 2 * 3 * \dots * N$

Algorithme	Programme PASCAL
<pre> Algorithme exo2_2; Variables N, i, P: entier ; Début {-*-*- Entrées -*-*-} Ecrire('Donner la valeur de N : '); Lire(N); {-*-*- Traitement -*-*-} P ← 1; Pour i ← 1 à N faire P ← P * i; Fin-pour; {-*-*- Sortie -*-*-} Ecrire('P =', P); Fin. </pre>	<pre> Program exo2_2; Var N, i, P: integer ; Begin {-*-*- Entrées -*-*-} Writeln('Donner la valeur de N : '); Readln(N); {-*-*- Traitement -*-*-} P:=1; for i:=1 to N do P:= P * i; {-*-*- Sortie -*-*-} Write('P =', P); End. </pre>

3) Calculer la somme $S = X + X^2 + X^3 + \dots + X^N$

Algorithme	Programme PASCAL
Algorithme exo2_3; Variables N, i: entier; X, S, P: réel; Début {--*-- Entrées --*--} Ecrire ('Donner la valeur de N : '); Lire (N); Ecrire ('Donner la valeur de X : '); Lire (X); {--*-- Traitement --*--} S ← 0; P ← X; Pour i ← 1 à N faire S ← S+P; P ← P*X; Fin-pour ; {--*-- Sortie --*--} Ecrire (' S = ', S:0:3); Fin.	Program exo2_3; Var N, i: integer; X, S, P: real; Begin {--*-- Entrées --*--} Write ('Donner la valeur de N : '); Read (N); Write ('Donner la valeur de X : '); Read (X); {--*-- Traitement --*--} S:=0; P:=X; for i:=1 to N do Begin S:= S+P; P:= P*X; End ; {--*-- Sortie --*--} Write (' S = ', S:0:3); End.

4) Calculer la somme $S = X + \frac{X^3}{2!} + \frac{X^5}{4!} + \frac{X^7}{6!} + \dots$ ($N^{\text{ème}}$ terme)

Algorithme	Programme PASCAL
Algorithme exo2_4; Variables N, i, F : entier; X, P, S : réel; Début {--*--*-- Entrées --*--*--} Ecrire('Donner la valeur de N : '); Lire(N); Ecrire('Donner la valeur de X : '); Lire(X); {--*--*-- Traitement --*--*--} S ← 0; P ← X; F ← 1; Pour i:=0 à (N-1) faire S ← S + P/F; P ← P*X*X; F ← F*(2*i+1)* (2*i+2); Fin-Pour ; {--*--*-- Sortie --*--*--} Ecrire(' S = ', S:0:3); Fin.	Program exo2_4; Var N, i, F : integer; X, P, S : real; Begin {--*--*-- Entrées --*--*--} Write('Donner la valeur de N : '); Read(N); Write('Donner la valeur de X : '); Read(X); {--*--*-- Traitement --*--*--} S:=0; P:=X; F:=1; For i:=0 to (N-1) do Begin S:= S + P/F; P:= P * sqr(X); F:=F*(2*i+1)* (2*i+2); End ; {--*--*-- Sortie --*--*--} Write(' S = ', S:0:3); End.

II.10. Exercices supplémentaires

Exercice 01 :

Réaliser les conversions suivantes :

$$2021 = (?)_2$$

$$(753)_8 = (?)_2$$

$$(10110110001)_2 = (?)_{10}$$

$$(101110011100011)_2 = (?)_8 = (?)_{16}$$

$$(753)_8 = (?)_{10}$$

$$(AB0793)_{16} = (?)_8$$

Exercice 02 :

a) Traduire les expressions suivantes en langage Pascal : $y = x^2 + \sqrt{\frac{|2x| + \sqrt{x}}{2e^x}}$; $z = e^{\sqrt{5x+|-3x|}}$

b) Définir les opérateurs **DIV** et **MOD** en donnant deux exemples numériques pour chacun.

Exercice 03 :

Soit a, b, c, d, x, y des variables réelles, tel-que : a=1, b=2, c=3, d=6

Évaluer les expressions suivantes en indiquant l'ordre d'évaluation :

$$(a + b) + (c + a * (d / 3)) + 6 / c + 2 * a$$

$$(a + b) < (c + a * (d / 3)) + 6 / c + 2 * a$$

$$(a > b) \text{ And Not } (c + a > d / 3) \text{ OR } (6 \text{ Mod } c = 2 \text{ Div } c)$$

Exercice 04 :

En PASCAL, indiquer, parmi cette liste de mots, les identificateurs valides et non-valides :

12K, a, x1, k12, prix unitaire, qte-stock, sinon, while, begin, hateur, largeur

Exercice 05 :

Ecrire un algorithme puis la traduction en Pascal d'un programme **Surface_Rectangle**, qui calcule la surface d'un rectangle de dimensions données et affiche le résultat sous la forme suivante : "La surface du rectangle dont la longueur mesure m et la largeur mesure m, a une surface égale à mètres carrés".

Exercice 06 :

Ecrire un algorithme puis la traduction en Pascal d'un programme **Trapeze**, qui lit les dimensions d'un trapèze et affiche sa surface.

Exercice 07 :

Ecrire un algorithme puis la traduction en Pascal d'un programme qui lit une **température** en degrés Celsius et affiche son équivalent en Fahrenheit.

Exercice 08 :

Exécuter les séquences d'instructions suivantes manuellement et donner les valeurs finales des variables A, B, C et celles de X, Y, Z.

a) $A \leftarrow 5$; $B \leftarrow 3$; $C \leftarrow B+A$; $A \leftarrow 2$; $B \leftarrow B+4$; $C \leftarrow B-2$

b) $X \leftarrow -5$; $Y \leftarrow 2*X$; $X \leftarrow X+1$; $Y \leftarrow \text{sqr}(-X-Y)$; $Z \leftarrow \text{sqr}(-X+Y)$; $X \leftarrow -(X+3*Y)+2$

Ecrire les algorithmes correspondants puis les programmes en Pascal correspondants et les exécuter.

Exercice 09 :

Ecrire un algorithme permettant d'effectuer une permutation circulaire de trois nombre entiers a, b et c.

Exemple : a=10, b=20 et c=30

Après permutation : a=30, b=10 et c=20

Exercice 10 :

Ecrire un programme permettant de lire la valeur de la température de l'eau et d'afficher son état :

« Glace » Si la température ≤ 0 ,

« Liquide » Si $0 < \text{la température} < 100$,

« Vapeur » Si la température ≥ 100 .

Exercice 11 :

Soit un service d'impression qui établit le prix d'impression d'une page selon le nombre de pages (nb_pages) :

a- Si nb_pages est inférieure ou égale à 10 : 5 D.A.

b- Si nb_pages est entre 11 et 20 : 4.5 D.A.

c- Si nb_pages est entre 21 et 60 : 3 D.A.

d- Si nb_pages est supérieure à 60 : 2.5 D.A.

Exercice 12 :

Ecrire un programme qui permet de résoudre l'équation du second degré $ax^2 + bx + c = 0$

Exercice 13 :

On demande d'écrire l'algorithme d'une fiche de paie journalière d'un ouvrier rémunéré à la tâche. Pour cela, on donne :

- La valeur de cette rémunération par pièces réalisées VP,
- Le salaire brut (SB) est calculé selon le nombre de pièces correctes réalisées pendant la journée (NPC) comme suit :

Si $NPC \leq 100$, l'ouvrier touche $NPC * VP$

Si $NPC > 100$, l'ouvrier touche $150 * VP$

- On enlève à la fin 10% du salaire pour les charges sociales (CS).

Calculer et afficher le salaire journalier brut (SB), les charges sociales (CS) et salaire journalier net (SN).

NB : Salaire brut=salaire totale ; Salaire net=salaire sans les charges sociales.

Exercice 14 :

Ecrire un programme Pascal qui calcule et affiche la **somme** et le **produit**, des 20 premiers entiers (de 1 à 20).

Exercice 15 :

Ecrire un programme Pascal faisant calculer et afficher le **factoriel** d'un entier naturel N donné. Sachant que (pour $N > 0$) : $N! = N \times (N-1) \times (N-2) \times \dots \times 3 \times 2 \times 1$.

Exercice 16 :

Ecrire un algorithme permettant de calculer la somme de tous les nombres impairs entre deux valeurs N et M.

Exercice 17 :

Ecrire un algorithme permettant de tester si un nombre N est premier en utilisant une boucle **Tant que** puis une boucle **Répéter**.

Exercice 18 :

Ecrire un programme Pascal qui permet de calculer la somme S suivante : (avec X un nombre réel donné)

$$S = \frac{X^2}{2} - \frac{X^4}{4} + \frac{X^6}{6} - \frac{X^8}{8} + \frac{X^{10}}{10} - \frac{X^{12}}{12} + \dots$$

Exercice 19 :

Ecrire un programme Pascal qui permet d'afficher tous les multiples de 7 positifs strictement et inférieur à 100. Le programme doit aussi calculer et afficher la somme, le produit et la moyenne de ces multiples de 7.

On veut aussi avoir sur l'écran à l'exécution du programme, les affichages sous la forme suivante :

Les multiples de 7 inférieurs à 100 sont : 7 14 21 La somme de ces nombres est : Le produit de ces nombres est : La moyenne de ces nombres est :
--