Exercices

1. Ecrire un algorithme qui calculer la somme de deux entiers (puis traduit le en programme pascal)?
2. Ecrire un algorithme qui permute entre deux entiers (puis traduit le en programme pascal)?
3. 2. Ecrire un algorithme qui déterminer si le milieu acide si le ph<7 et milieu base si le ph>7(puis traduit le en programme pascal)?
3. Ecrire un algorithme qui résoudre une équation de 2 ordre et donner les différentes solutions (puis traduit le en programme pascal)?
4. Ecrire un programme pascal qui reçoit une valeur et calculer la somme de 1 jusqu'à cette valeur (avec deux méthodes) (puis traduit le en programme pascal)?
5. Ecrire un algorithme qui calculer le factorielle de la valeur N donne (N>0) (puis traduit le en programme pascal) ?
6. Ecrire un algorithme qui accepter qu'une valeur négative (puis traduit le en programme pascal)?

**********Cours**********

Algorithme**<algo1>* *<Déclarations>* ***Début *<Corps>* ***Fin.***	***Program** prog1;* ***uses** wincrt;* *<déclarations>;* ***Begin*** *<Instructions>;* ***End.***	***Entête :** on déclare le nom de l'algorithme à travers un identificateur.* ***Déclarations :** on déclare toutes les données utilisées par l'algorithme* ***Corps :** représente la séquence d'actions (instructions)*

Types de données :

Algo	*PASCAL*	
Entier	*Integer*	*{…, 4, 3, 2, 1, 0, 1, 2, 3, 4, ...}*
Réel	*Real*	*[-ω, +ω]*
Booléen	*Boolean*	*TRUE FALSE*
Caractère	*Char*	*{a, b,…, z, A, B, …, Z, _ }*
chaîne	*String*	*une séquence d'un ou plusieurs caractères*

Déclarations

on déclare toutes les données d'entrées et de sorties sous forme de constantes et de variables.
*– **Constantes :** sont déclarées comme suit :*
CONST <identificateur> = <valeur>;

Exemples :

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 : Les variables sont déclarées comme suit :
***VAR** <identificateur> : <type>;*
On a cinq types de données de base :
– Entiers
– Réels
– Caractères
– Chaîne de caractères
– Booléens, contenant deux valeurs : True ou False ;

Exemple

Algorithme	***Programme***
***var** x : reel*	***var** x:real;*
***var** n, m : entier*	***var** n, m:integer;*
***var** s : chaîne*	***var** s:string;*
***var** b1, b2, b3 : booleen*	***var** b1, b2, b3:boolean;*
***var** c1 : caractere*	***var** c1:char;*

Corps

Le corps d'un algorithme est constitué d'un ensemble d'actions / instructions séquentiellement et logiquement ordonnées. Les instructions sont de cinq types, à savoir :
*– **Lecture :** L'opération de faire entrer des données à l'algorithme. Une lecture consiste à donner une valeur arbitraire à une variable.*
*– **Écriture :** L'opération d'affichage de données. Elle sert à afficher des résultats ou des messages.*
*– **Affectation :** ça sert à modifier les valeurs des variables.*
*– **Structure de contrôle :** Elle permet modifier la séquentialité de l'algorithme, pour choisir un chemin d'exécution ou répéter un traitement.*
*– Structure de **Test alternatif simple / double (si)***
*– Structure répétitives (itérative) **(répété)***
*– la boucle **Pour***
*– la boucle **Tantque***
*– 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.

Types d'instructions

Instructions اوامر d'Entrées/Sorties (Lecture / Écriture)

Instructions Entrées (Lecture)

Une instruction d'entrée nous permet dans un programme de donner une valeur quelconque à une variable. Ceci se réalise à travers l'opération de lecture. La syntaxe نحو et la sémantique مدلولي d'une lecture est comme suit:

Algorithme	***PASCAL***	***Signification***
Lire(a)	***Read(a);*** ***readln(a);***	*Donner une valeur quelconque à la variable dont l'identifiant <a>.*
Lire(x1,x2,…);	***read(x1,x2,…);***	*Donner des valeurs aux variables x1 x2, ...etc.*

Instructions Sorties (Écriture)

Une instruction de sortie nous permet dans un programme d'afficher un résultat (données traitées) ou bien un message (chaîne de caractères). Ceci se réalise à travers l'opération d'écriture.
La syntaxe et la sémantique d'une écriture est comme suit :

Algorithme	***PASCAL***	***Signification***
Ecrire(v1|PI |10| a+b)	***write(v1|PI |10| a+b);*** ***writeln(v1|PI |10| a+b);***	*Afficher une valeur d'une variable, d'une constante, valeur immédiate ou calculée à travers une expression.*

***Exemples** :*
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 de la somme de 5 et 2 : afficher 7}
écrire(a+b-c) write(a+b-c); {afficher le résultat de l'expression arithmétique : a+b-c}
écrire(5<2) write(5<2); {afficher le résultat de la comparaison 5 < 2, le résultat est la valeur booléenne FALSE}
Exemple
x :=5 ;
écrire('La valeur de x : ', x) write('La valeur de x :', x) ;
*{Afficher sur l'écran « **la valeur de x :** 5 »}.*

Instruction d'affectation

Une affectation consiste à donner une valeur (immédiate, constante, variable ou calculée à travers une expression) à une variable. La syntaxe d'une affectation est :
a ← 5 | y | x+y
a :=5 | y | x+y;

Exemples :

a ← 5	*a:=5;*	*{mettre la valeur 5 dans la variable a}*
b ← a+5	*b:=a+5;*	*{mettre la valeur de l'expression a+5 dans la variable B}*
sup ← a>b	*sup:=a>b;*	*{a>b donne un résultat booléen, donc sup est une variable booléenne}*

***Structures بنية هيكلة de contrôles :** permettent de choisir entre deux ou plusieurs chemins d'exécution تنفيذ (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'exécutions ou refaire un traitement plusieurs fois. Les structures de contrôle sont de deux types : Structures de contrôles conditionnelles شرطي et structures de contrôle répétitives تكراري (itératives).*

***Structures de contrôle conditionnel :** Ces structures sont utilisées pour décider de l'exécution d'un bloc d'instruction : est-ce-que ce bloc est exécuté ou non. Ou bien pour choisir entre l'exécution de deux blocs différents. Nous avons deux types de structures conditionnelles :*

***Test alternatif simple :** Un test simple 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 ne l'exécute pas. La syntaxe d'un test alternatif simple est comme suit :*

si** <Condition> **alors *<instruction(s)>* ***finsi**;*	***if** <condition> **then*** ***begin*** *<instruction(s)>;* ***end**;*

Exemple :

lire(x) ***si** x > 2 **alors*** *x ← x + 3* ***finsi*** *écrire (x)*	*read(x);* ***if** x > 2 **then*** ***begin*** *x:= x + 3;* ***end**;* *write(x);*

***Test alternatif double :** Un test double contient deux blocs d'instructions : on est amené à décider entre le premier bloc ou le second. Cette décision est réalisée selon une condition (expression logique ou booléenne) qui peut être vraie ou fausse. Si la condition est vraie on exécute le premier bloc, sinon on exécute le second. La syntaxe d'un test alternatif simple est :*

si <Condition> alors <instruction(s)1> sinon <instruciton(s)2> finsi	if <condition> then begin <instruction(s)1>; end else begin <instruction(s)2>; end;
Exemple :	

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

Structures de contrôle répétitives : *Les structures répétitives nous permettent de répéter un traitement un nombre fini de fois. Nous avons trois types de structures itératives (boucles) :*

1.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 **p&&our** est comme suit :*

pour *i←x1* ***à*** *n* ***faire*** <instruction(s)> ***finPour***;	***for*** *i:=x1* ***to*** *n* ***do*** ***begin*** <instruction(s)>; ***end***;
i : variable entière x1 : valeur initiale n : valeur finale *(voir exercice 4&5)*	

2.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 :*

tant-que <condition> ***faire*** <instruction(s)> ***finTant-que***;	***while*** <condition> ***do*** ***begin*** <instruction(s)>; ***end***;
<condition> : expression logique qui peut être vraie ou fausse. *(voir exercice 4&5)*	

*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**.*

*Toute boucle **pour** peut être remplacée par une boucle **tantque**, cependant l'inverse n'est pas toujours possible.*

. 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 :*

repéter <instruction(s)> ***jusqu'à*** <condition>;	***repeat*** <instruction(s)>; ***until*** <condition>;
<condition> : expression logique qui peut être vraie ou fausse *(voir exercice 6)*	

*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 n'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** et 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.*

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 :

aller à <num_etiq> ... <num_etiq> : . . .	***goto*** <num_etiq>; ... <num_etiq> : . . .

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 :

algorithme *branchement* ***variables*** *a, b, c : entier ;* ***début*** *lire (a, b);* *2: c ← a;* ***si*** *(a > b)* ***alors*** ***aller à*** *1;* ***finsi*** *a ← 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.***

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 à appartient au bloc si.*

Les opérateurs

On trois type d'opérateurs : Opérateurs arithmétiques, opérateurs relationnels (de comparaison) et opérateurs logiques.

Les opérateurs arithmétiques

+**(Addition),**-** (Soustraction), ** (Multiplications), / (Division), **div** (Division entière, elle fournit la partie entière d'une division), **mod** (Modulo ou reste de division).*

Si les deux opérandes sont de type entier, le résultat est de type entier. Soit :

entier + entier = entier	***entier – entier = entier***
entier * entier = entier	***entier / entier = réel.***
entier DIV entier = entier	***entier MOD entier = entier***

Les opérateurs relationnels

On distingue les opérateurs de relation suivants :

= (égale), <> (différent), < (inférieur), > (supérieur), <= (inférieur ou égale), >= (supérieur ou égale)

Appliqués aux opérandes, ils fournissent une résultat booléens. Soit :

12 = 5 fournit le résultat (faux) ou (false) en anglais

12 = 12 fournit le résultat (vrai) ou (true) en anglais

45 < 49 fournit le résultat (vrai)

Les opérateurs logiques

On distingue les opérateurs logiques suivant : AND, OR et NOT.

Il s'applique à des opérandes de type booléen, et fournissent une valeur de type booléen (logique). Les opérateurs AND et OR sont des opérateurs binaires, c'est-à-dire qu'ils s'appliquent à deux opérandes. Par exemple, on écrit :

opérande1 AND opérande2

L'opérateur NOT est un opérateur unaire, c'est-à-dire qu'il s'applique à un seul opérande. Soit :NOT opérande.

Exemple :

(45 > 59) AND (15 = 15) → faux AND vrai → Résultat : faux (FALSE)

(25 > 45) OR (47 < 50) → faux OR vrai → Résultat : vrai (TRUE)

NOT (25 > 45) → NOT (Faux) → Résultat : vrai (TRUE)

Les fonctions

Les fonctions standards appliquées aux entiers ou réels

Fonction	***L'appel avec paramètre***	***Le résultat retourné***
ABS	*ABS(x)*	*Retourne la valeur absolue d'un nombre x*
EXP	*EXP(x)*	*Retourne l'exponentiel d'un nombre x*
LN	*LN(x)*	*Retourne le logarithme népérien d'un nombre x*
LOG	*LOG(x)*	*Retourne le logarithme à base 10 d'un nombre x*
SQRT	*SQRT(x)*	*Retourne la racine carrée d'un nombre x*
SQR	*SQR(x)*	*Retourne le carré d'un nombre x*
Arctan	*Arctan(x)*	*Retourne l'arc tangente d'un nombre x*
Cos	*Cos(x)*	*Retourne le cosinus d'un nombre x*
Sin	*Sin(x)*	*Retourne le sinus d'un nombre x*
Round	*Round(x)*	*Retourne la valeur arrondie d'un nombre x*
Trunc	*Trunc(x)*	*Retourne la partie entière d'un nombre x*

Règles d'évaluation des expressions arithmétiques

Règle 1 : *On évalue d'abord le contenu des parenthèses en commençant par les parenthèses les plus internes.*

Règle 2 : *On commence à effectuer les opérateurs de plus haute priorité. Dans le cas des opérateurs de même priorité, on commence par le plus à gauche.*

Règles de priorités des opérateurs

La priorité des opérateurs arithmétiques et logiques est dans l'ordre suivant :

1) Les parenthèses

2) Les fonctions

3) Le moins unaire, le NOT

*4) *, /, DIV, MOD, AND*

5) +, , OR

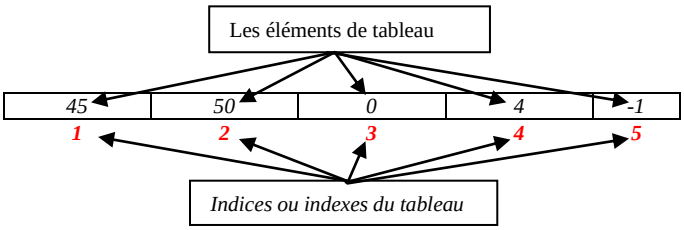
6) =, <>, <, >, <=, >=

Exercices :

1- Ecrire un programme qui permet de faire la lecture et l'écriture d'un tableau de N éléments
2- Ecrire un programme qui permet de faire la somme de deux vecteurs V1 et V2
3- Ecrire un programme qui permet de faire le produit scalaire de deux vecteurs V1 et V2
4- Ecrire un programme qui permet de chercher un élément **val** dans un tableau à une dimension (vecteur). S'il existe on récupère son rang.
5- Ecrire un programme qui permet de faire la lecture et l'écriture d'un tableau bidimensionnel de NxN éléments.
6- Ecrire un programme qui permet de faire le produit de deux tableaux bidimensionnel.

Variables Indicées – Le type Tableau

Pour représenter des tableaxu, soit les vecteurs ou les matrices, on doit utilisé les ***variables indicées ou indexées****.*



1- Les tableaux unidimensionnels (Vecteurs) : *Les tableaux à une seule dimension correspondent aux tableaux avec une seule plage d'indices.*
C'est-à-dire, pour accéder à une composante on a besoin d'un seul indice (valeur entière). Autrement dit : les cases sont repérées avec un seule indice qui représente une valeur entière (valeur immédiate, constante ou variable entière, ou expression donnant un résultat entier).

3.2.1. Accès à un élément du tableau

En programmation un élément de tableau est désigné par le nom du tableau suivi de l'indice (la position) de l'élément dans le tableau. Soit :

<Nom du Tableau>[<Indice de l'élément>]

Exemple :

Le tableau sous dessus :
V[1]=45, V[2]=50, V[3]=0, V[4]=4, V[5]=-1.

Déclaration d'un tableau unidimensionnel

La syntaxe à suivre pour déclarer un tableau est la suivante :

Syntaxe Algorithmique	***Syntaxe en PASCAL***
constante *MAX = <valeur_entière>* ***variable*** <id_tab>:Tableau[1..MAX] de <type>	***const*** *MAX = <valeur_entière>;* ***var*** <id_tab>:***Array*** *[1..MAX]* ***of*** *<type>;*
Exemple : *V est un tableau de 50 composantes réelles (c'est comme si on a déclaré 50 variables réelles)*	
constante *MAX = 50* *variable V:Tableau[1..MAX] de réel;*	*const* *MAX = 50;* *var V:Array [1..MAX] of real;*

Remarques :

✗ *C'est pas obligatoire d'utiliser une constante pour la taille maximale de tableau ; cependant, c'est une bonne pratique dans la programmation*

✗ *La plage d'indice 1 .. MAX représente l'indice du premier élément 1 et l'indice du dernier élément MAX.*

✗ *C'est pas obligatoire d'utiliser toutes les composantes du tableaux, pour cela on déclare une variable entière n qui représente la taille du tableau à utiliser. La valeur de cette variable sera introduite au cours de l'exécution (par lecture)*

Initialisation d'un tableau unidimensionnel

Un tableau peut être initialisé en utilisant deux méthodes :
– En utilisant des affectations ;
– En utilisant la lecture à partir d'un fichier de données ou à partir de clavier.

a) Par affectations : *On utilise pour cela l'opération d'affectation.*

Exemple :

V[1] := 45; V[2] := 50; V[3] := 0; V[4] := 4; V[5] := -1;

b) Par lecture : *On utilise pour cela l'opération de lecture READ : C'est la méthode la plus commode.* ملائم

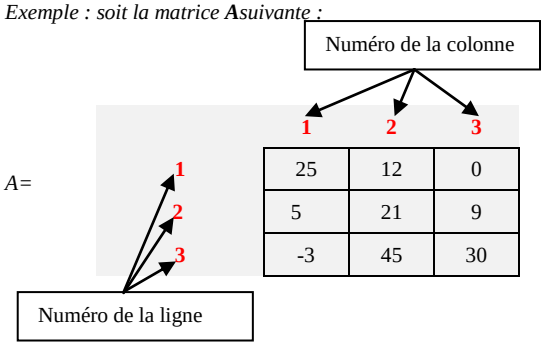
Exemple1 : *Exemple d'algorithme / programme Pascal permettant de lire et d'écrire (afficher) le tableau précédent :*

Algorithme	***Programme en Pascal***
algorithme *LireEcrireTableau* ***variables*** *v:tableau[1..5] de réel* *i:entier* ***Début*** *écrire('Introduire V :')* ***pour*** *i:=1* ***à*** *5* ***faire*** *lire(v[i])* ***finPour*** *écrire('Affichage V :')* ***pour*** *i:=1* ***à*** *5* ***faire*** *écrire(v[i])*	***program*** *LireEcrireTableau;* ***uses*** *wincrt;* ***var*** *V:****Array****[1..5]* ***of*** *real;* *i:integer;* ***Begin*** *writeln('Introduire V : ');* ***for*** *i:=1* ***to*** *5* ***do*** *Read (V[I]);* *writeln ('Affichage V : ') ;* ***for*** *i:=1* ***to*** *5* ***do*** *Write (V[I]);*
finPour ***Fin***	***End****.*

2-Les tableaux bidimensionnels (Matrices)

Un tableau est dit bidimensionnels (ou à deux dimensions), si chacun des ses éléments (composant) est repéré par un couple d'indices (i, j) où i est le numéro de la ligne et j est le numéro de la colonne où l'élément est situé. Ce type de tableau est appelé ***Matrice****.*

Exemple : soit la matrice ***A****suivante :*



Déclaration de tableaux bidimensionnels

Pour déclarer la matrice A définie précédemment, on a deux façons :

1ère Façon :	
ALGORITHME	***Programme en PASCAL***
Variables *A:****Tableau*** *[1..3,1..4]* ***de*** *Réel*	***Var*** *A:****Array****[1..3,1..4]* ***of*** *real;*
2ème Façon :	
Type *Ligne =* ***Tableau*** *[1..4]* ***de*** *réel* ***Variables*** *A:****Tableau*** *[1..3]* ***de*** *Ligne*	***Type*** *Ligne =* ***Array****[1..4]* ***of*** *real;* ***Var*** *A:****Array****[1..3]* ***of*** *Ligne;*