UNIVERSITE DE BATNA 1
HADJ LAKHDAR

Faculté des Sciences Economiques, Commerciales et Sciences de Gestion

Support de cours

MODULES: Informatique I & II

Intitulé: ALGORITHMIQUE I & II

2ième année LMD
Spécialité: Economie & Commerce

2017-2018

Réalisé par: Dr. Leïla BOUSSAAD

Table des Matières

Chapitre 1

Introduction à l'algorithmique et à la programmation

Sommaire

1.1 Définitions

1.1.1 Algorithme

Un algorithme est une suite finie d'instructions nécessaires permettant de résoudre un problème particulier.

- Un algorithme n'est pas un programme.
- Un algorithme est indépendant du langage dans lequel il est implémenté, et de la machine qui exécutera le programme correspondant.
- Il existe plusieurs algorithmes pour résoudre un problème posé.
- Le "langage algorithmique" est un compromis entre un langage naturel et un langage de programmation.

1.1.2 Programme informatique

Un programme informatique est un algorithme traduit dans un langage de programmation(exemple : Pascal, C, C++, Java,..). Un programme doit être exécuté pour effectuer le traitement souhaité.

- Un programme informatique peut être défini comme une suite d'instructions destinées à être exécutées par un ordinateur.
- Une instruction (ordre) dicte à l'ordinateur l'opération nécessaire qu'il doit effectuer.

1.1.3 Programmation

La programmation est la conception d'un algorithme (analyse du problème, le choix de méthode de résolution et le choix des représentations de données) et sa traduction dans un langage de programmation pour être exécuté par une machine.

1.1.4 Langage de programmation

Un langage de programmation est un ensemble de termes et de règles de vocabulaire et de grammaire compréhensible par un ordinateur, qui permet de rédiger un programme informatique.

- Un programme écrit dans un langage de programmation est appelé un programme source (ou code source)
- Un programme écrit dans le langage machine est appelé un programme exécutable.
- Un code source ne peut être directement exécuté par la machine.
- Pour être exécuté par une machine, un programme source doit être traduit en langage machine
- Suivant le langage utilisé, un programme peut être interprété ou compilé

Langage interprété: Un programme écrit dans un langage interprété a besoin d'un programme auxiliaire (l'interpréteur) pour traduire au fur et à mesure les instructions du programme.

Langage Compilé: Un programme écrit dans un langage compilé va être traduit une fois pour toutes par un programme annexe, appelé compilateur, afin de générer un nouveau fichier contenant des instructions compréhensibles par la machine (c.-à-d. par le micro processeur), on dit d'ailleurs que ce fichier est exécutable.

Langage intermédiaire: Certains langages sont en quelque sorte compilés et interprétés, car le programme écrit avec ces langages peut subir une phase de compilation intermédiaire vers un fichier écrit dans un langage qui est différent du programme source et non exécutable (nécessité d'un interpréteur).

1.1.5 Langage Pascal

Le Pascal est un langage de programmation compilé, il tient son nom du mathématicien français Balaise Pascal, il a été conçu au début des années 70 par N. Wirth, il est l'un des langages de programmation les plus utilisés dans le milieu scolaire pour enseigner les notions de base de la programmation , car c'est un langage facile à apprendre et il possède des instructions très claires et bien structurées.

1.1.6 Comment programmer en Pascal?

1. Edition: Un programme source Pascal peut être écrit par n'importe quel éditeur de texte ou dans l'éditeur dédié de l'environnement de programmation. Le programme obtenu est stocké dans un fichier avec l'extension (.pas).

2. Compilation: le résultat de cette étape s'appelle le code objet du programme, le code objet ne peut être crée que si le code source ne contient pas d'erreurs syntaxiques. Dans cette étape, toute erreur de nature syntaxique est localisée par le compilateur.

   Remarque Un programme qui est syntaxiquement correct, ne veut pas dire qu'il est sémantiquement correct (c.-à-d. il fait ce qu'on attend de lui).
   Pour s'assurer qu'un programme est sémantiquement correct, il faut l'exécuter plusieurs fois, dans les différentes situations possibles, surtout dans les cas particuliers.

3. Edition de liens: elle est réalisée par un programme auxiliaire, qui s'appelle éditeur de liens et qui intègre dans le fichier objet le code machine de toutes les fonctions utilisées dans le programme et non définies dans le fichier source. L'éditeur de liens génère un fichier exécutable (avec l'extension .exe) qui peut être chargé en mémoire pour être exécuté.

1.1.7 Environnement de programmation intégré

Un environnement de programmation intégré permet d'éditer un programme, de le compiler et de l'exécuter sans passer d'un programme à un autre, le code source et le code exécutable peuvent rester tous les deux en mémoire.
Turbo pascal est un environnement de programmation intégré, il est évolué sans cesse.

Ce cours concerne l'environnemnt Borland Pascal for Windows (BPW), même tous les exemples fournis dans ce document sont écrits dans cet environnement BPW.

1.1.8 Quelques références Bibliographiques

- L'aide de l'environnement de programmation (BPW).
- C. Delannoy, Programmer en Turbo Pascal, Edition Eyrolles, 2002.
- M. C. Belaid, Algorithmique et programmation en Pascal, cours, exercices et travaux pratique avec corrigés, Edition Les Pages Bleues, 2004.
- M. ZAIR, Apprendre l'algorithmique, cours et exercices corrigés, Edition Les Pages Bleues, 2016.
- https://pascal.developpez.com/

Chapitre 2

Les éléments de base du language PASCAL

Sommaire

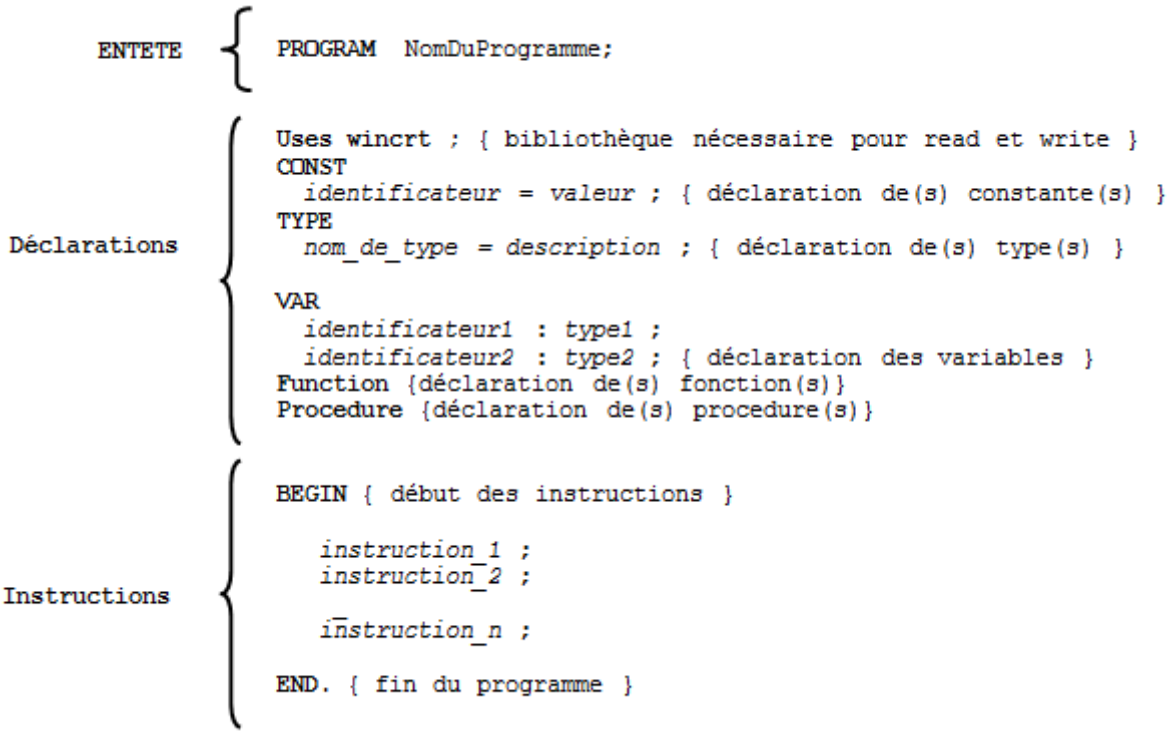
2.1 Introduction

Un programme exécute des instructions sur des données pour résoudre un problème, et fournit des résultats, donc le programmeur doit:

- Prévoir de l'espace mémoire pour stocker ces données et ces résultats, c'est ce qu'on appelle en programmation: la déclaration de variables.
- Permettre l'entrée des données.
- Spécifier les instructions à exécuter.
- Permettre l'affichage des résultats.

2.2 Structure d'un programme écrit en langage pascal

La structure d'un programme PASCAL est définie selon la figure suivante:



En première analyse, on trouve:

- Le mot réservé program suivi d'un identificateur (le nom du programme), suivi d'un point-virgule,
- les différentes déclarations dont on a besoin (types, constantes, variables, fonctions et procédures),

- le bloc d'instructions encadré par les mots réservés begin et end,
- et finalement un point,
- les instructions sont séparées par des ' ;' .

Remarques

1. En général, le point-virgule (;) est utilisé pour séparer les instructions, tandis que la virgule (,) est utilisée pour séparer les éléments d'une liste,
2. il est possible de mettre plusieurs instructions sur une même ligne, mais il est recommandé de n'en écrire qu'une seule instruction par ligne,
3. ce qui est entre accolades { } constitue un commentaire,
4. tout ce qui suivra le point final sera ignoré par le compilateur.

2.3 Les commentaires dans un programme

Les commentaires sont des éléments du texte d'un programme qui sont ignorés par le compilateur. Ils servent à donner des informations sur le programme. En Pascal, on peut écrire des commentaires de deux façons:

- Commentaires sur une seule ligne commencés par: //.
- Commentaires sur plusieurs lignes encadrés par {et}, ou par (* et *).

2.4 Exemples de programmes écrits en Pascal

Exemple 1

Prenons comme premier exemple un programme PASCAL qui va afficher sur l'écran la somme de deux nombres obtenus par une lecture au clavier:

ENTETE

```
1  PROGRAM Addition;
```

Section déclarative

```
1  Uses wincrt ;
2  VAR
3    Somme : INTEGER;
4    Nombre1, Nombre2 : INTEGER;
```

Bloc d'instructions

```
1  BEGIN
2    Writeln ('Donnez un nombre : ');
3    Readln (Nombre1); { Lecture du premier nombre }
4    Writeln ('Donnez un deuxième nombre : ');
5    Readln (Nombre2); { Lecture du 2ème nombre }
6    Somme := Nombre1 + Nombre2;
7    Write ('La somme de ', Nombre1 , ' et ',Nombre2, ' = ', Somme);
8  END.
```

Exemple 2

Prenons un autre exemple, un programme PASCAL qui va afficher sur l'écran le Net à payer d'un salarié, calculé à partir du salaire brut obtenu par une lecture au clavier, et la valeur de l'IRG (Impôt sur revenu global), telque:

Net à payer = salaire brut * IRG/100 et l'IRG = 20.3%.

ENTETE

```
1  PROGRAM Net_a_payer;
```

Section déclarative

```
1  Uses wincrt ;
2  CONST
3    IRG = 20.3;
4  VAR
5    NET, BRUT  : real;
```

Bloc d'instructions

```
1  BEGIN
2    Writeln ('Donnez votre salaire brut : ');
3    Read (BRUT); { Lecture du salaire brut }
4    NET := Brut*IRG/100;
5    Write('Le net à payer est:', NET);
6  END.
```

2.5 Les expressions

Il existe deux types d'expressions, celles de déclaration et celles de traitement.

1. Les expressions de déclaration permettent de préciser les données nécessaires pour

résoudre le problème, ainsi que leur type, elles se trouvent dans la partie déclaration du programme.

2. Les expressions de traitement permettent de décrire les traitements exécutés sur les données pour obtenir les résultats du problème, elles se trouvent dans la partie instructions du programme.

2.6 Les identificateurs

Les identificateurs sont les noms utilisés pour nommer (identifier) des objets dans un programme, qui peuvent être des variables, des constantes, ou des sous-programmes (procédures et fonctions).
En Pascal, les identificateurs doivent commencer par une lettre et peuvent être poursuivi par n'importe quelle combinaison de lettres (A-Z, a-z), chiffres (0-9) ou blancs soulignés (_)[Shift+ 8].

Remarques

- Vous pouvez utiliser n'importe quel mot (non réservé)qui ne contient ni espace, ni accent, ni apostrophe, et un chiffre ne peut pas être placé au début.
- Aussi les caractères @, x, &, #, +, −, ∗,et / sont interdits:
- Le langage Pascal ne fait aucune distinction entre lettre minuscule et lettre majuscule.
- Certains identificateurs sont des mots réservés du langage, et ne peuvent donc pas être utilisés pour désigner des variables ou autres entités.

2.6.1 Exemples d'identificateurs corrects et incorrects

- **Identificateurs corrects:**
  Amine - BlackBird - RS232 - Au_Debut - Count
- **Identificateurs incorrects:**
  1. **Au-Dela:** Le signe (-) est un opérateur arithmétique, il désigne, dans Pascal, l'opération de soustraction.
  2. **7eme:** Un identificateur doit commencer par une lettre ou un blanc souligné (_).
  3. **SYS$READ:** Les caractères spéciaux ne sont pas permis.
  4. **Pas a pas:** Un identificateur ne doit pas contenir un espace.
  5. **Program:** Le mot Program peut apparaitre uniquement dans l'entête du programme (c'est un mot réservé du langage).

2.6.2 Les mots réservés du langage PASCAL

and	array	begin	case	const
div	do	downto	else	end
exit	false	file	for	function
if	of	implementation	in	interface
mod	new	not	to	or
procedure	program	record	repeat	then
true	type	until	uses	var
while	With			

2.7 Les constantes et les variables

2.7.1 Les constantes

Une constante est emplacement réservé en mémoire, qui contient la valeur d'une donnée non modifiable. Les constantes sont déclarées dans la partie des déclarations d'un programme, après le mot réservé **const**, suivant la syntaxe ci-dessous:

Syntaxe

```
1  Const
2  NomConstante = ValeurConstante;
```

Remarques

- On utilise le symbole = dans les déclarations de constantes.
- Il est possible de déclarer plusieurs constantes, sans répéter le mot clé **const**.

Exemple

```
1  Const
2   Pi=3.14 ;
3   LARG = 640;
4   HAUT = 480;
```

2.7.2 Les variables

Une variable est emplacement réservé en mémoire, qui contient la valeur d'une donnée modifiable. Chaque variable possède un nom unique (identificateur) par lequel on peut accéder à son contenu.

Exemple

On peut avoir en mémoire une variable X et une variable Y qui contiennent les valeurs 5 et 12 :

On dit : la variable X contient la valeur 5, et la variable Y contient la valeur 12.
Une variable est caractérisée par :

1. Un identificateur: le nom qui sert à repérer la variable.
2. Un type: format des données qu'elle contient (ce qui permet de prévoir la taille de l'espace mémoire qui leur est nécessaire)
3. Une valeur.

Remarques

- Il ne faut pas confondre le nom de la variable et sa valeur. Le nom est l'identificateur par lequel on peut accéder à sa valeur, alors que la valeur d'une variable est le contenu de celle-ci, qui peut être numérique, alphanumérique, booléen, ou autre type.
- la valeur d'une variable peut varier au cours du programme. L'ancienne valeur est tout simplement écrasée et remplacée par la nouvelle.
- Pour nommer une variable, il est recommandé de lui donner un identificateur qui reflète sa fonction.
- Une variable peut être :
    1. Donnée : variable nécessaire pour stocker les données d'entrée nécessaires pour faire fonctionner le programme.
    2. Résultat : Pour stocker les résultats fournis par le programme.
    3. Auxiliaire : qui sert à stocker des informations temporaires, à faire des calculs ou à faire fonctionner des structures de contrôle.

Déclaration de variables

La déclaration de variables permet de réserver de l'espace mémoire pour stocker des données, l'espace réservé dépend du type de ces données (exemple: des entiers, des réels, des caractères, etc).

- Les variables doivent être déclarées avant d'être utilisées dans le programme.
- La déclaration d'une variable indique deux choses à préciser:
    1. son identificateur (son nom) ;
    2. son type (pour connaître sa taille).

- Les variables sont déclarées dans la partie des déclarations d'un programme, après le mot réservé var, suivant la syntaxe ci-dessous:

  syntaxe

```
1   var
2   NomVariable : TypeVariable;
```

- Il est possible de déclarer plusieurs variables, sans répéter le mot clé var.
- Pour des variables de même type, on les écrit dans la même ligne séparées par des virgules.

  Exemple

```
1   Var
2   x : REAL;
3   s , t : STRING;
```

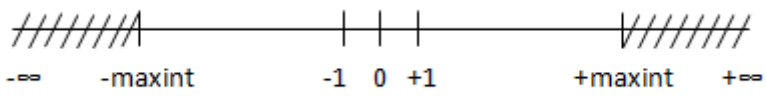
2.7.3 Les types de variables

Définition

Un type correspond à un ensemble de valeurs ainsi que les opérateurs possibles sur ses valeurs.

Les types prédéfinis

1. Le type entier: Ce type est désigné par l'identificateur prédéfini integer. Les valeurs de ce type forment un sous-ensemble de l'ensemble des nombres entiers, les bornes de ce sous ensemble dépendent de l'implémentation du compilateur. La valeur prédéfinie maxint définit la valeur absolue du plus grand nombre entier représentable. Le sous ensemble désigné par le type integer est donné par le schéma qui suit:



2. Le type réel: Ce type est désigné par l'identificateur prédéfini real. Même que le type integer, les valeurs de ce type forment un sous-ensemble de l'ensemble de nombres réels.
3. Le type booléen (logique): Ce type est désigné par l'identificateur prédéfini boolean. Deux valeurs possibles pour ce type : true (vrai) et false (faux).

4. Le type caractère: Ce type est désigné par l'identificateur prédéfini char. Les variables de ce type contiennent un caractère, et les valeurs de ce type sont en général ordonnées suivant l'ordre des codes internes des caractères sur l'installation considérée (code interne: ASCII,..). Un caractère doit être écrit entre deux quotes, ou le symbôle ♯ suivi de son code ASCII.

   Exemples

   - Les caractères lettres: 'A', 'B',..., 'a','b',...
   - Les caractères chiffres: '1','2',.....,'9'
   - Les caractères spéciaux: '.', '=', ';', '
   - Le caractère espace: ' '
   - Le caractère quote: '''' , la première quote pour dire qu'il va y avoir un caractère, les 2 suivantes qui symbolisent la quote (car une seule quote voudrait dire fin du caractère), la dernière signifie fin du caractère.

5. Le type chaîne de caractères: Ce type est désigné par l'identificateur prédéfini string. Les variables de ce type contiennent une suite de caractères(un mot ou une phrase). Celle-ci doit être tapée entièrement sur une seule ligne, et écrit entre deux quotes. Exemples:

   - 'algorithme', 'Cours de programmation','and', 'Langage Pascal',...
   - 'Cette chaîne de caractère est invalide parce qu'elle
     est tapée en deux ligne'

   Une chaîne de caractères peut être déclarée de deux manières, comme suit:

```
1  Var
2  CH : STRING;
```

   ou bien

```
1  Var
2  CH : STRING[n]; {n est une constante entière}
```

   La première déclaration concerne les chaînes de caractères pouvant contenir au plus 255 caractères, par contre la deuxième concerne les chaînes de caractères pouvant contenir au plus n caracères.

Nouveaux types

Il est possible de créer de nouveaux types. Les types définis par le programmeur sont appelés types manufacturés. Les plus simples sont le type énuméré et le type intervalle:

1. Le type énuméré: Pour définir ce type, il faut citer toutes ses valeurs, dans l'ordre, et il ne faut pas déclarer une valeur dans deux types différents.

   Syntaxe

```
1  Type
2  NomType =(Val1,Val2,...,Valn) ;
```

Exemple

```
1  Type
2          c_feux_t = (Rouge, Orange, Vert);
3  Var
4          feux : c_feux_t;
```

Ecriture incorrecte

```
1  Type
2          X=(samedi, dimanche, lundi) ;
3          Y=(dimanche, lundi, mardi) ;
```

2. Le type intervalle: C'est un sous-ensemble de valeurs successives d'un type ordinal(integer, char, boolean). Les valeurs de ce type sont comprises entre une valeur inférieure et une valeur supérieure, avec ces valeurs incluses.

Syntaxe

```
1  Type
2          NomType= Val_inf..Val_sup ;
```

Exemple

```
1  Type
2           T1 = 20..45 ; { Des valeurs entières successives}
3           T2= '0'..'9'; {Des valeurs successives de type char}
4  Var
5          x: T1; {x peut prendre les entiers de 20 à 45}
6          c: T2; { c peut prendre les caractères de '0' à '9'}
```

- le type real n'est pas ordinal, et donc on ne pas créer un type intervalle avec des réels, il n'y a pas de notion de réels successifs.
- Lorsqu'on crée un type T2 à partir d'un type T1, ce type T1 doit déjà exister ; donc T1 doit être déclaré avant T2.

2.8 Opérations sur les variables et les constantes

2.8.1 Opérateurs arithmétiques

Ils opèrent sur des variables numériques (entières ou réelles).

Opérateurs de signe	+ / -
Multiplication	*
Division entière	div
Division réelle	/
Modulo (reste de la division entière)	mod
Addition	+
Soustraction	-

Remarques
Si x et y sont deux entiers, alors:

- x/y: donne un résultat de type réel.
- x div y donne le dividende de la division entière de x par y, qui est de type entier.
- x mod y donne le reste de la division entière de x par y.
- Si y = 0, les opérateurs /, div, et mod produisent une erreur à l'exécution.

2.8.2 Opérateurs logiques

Ils opèrent sur les variables booléennes.

and	Et logique
or	Ou logique
not	Négation

Table de vérité des opérateurs: not, and, et or

A	B	not A	A and B	A or B
true	true	false	true	true
true	false	false	false	true
false	true	true	false	true
false	false	true	false	false

2.8.3 Opérateurs relationnels

>	Supérieur
>=	Supérieur ou égal
<	Inférieur
<=	Inférieur ou égal
=	Egal
<>	Différent

1. Le = et <> opèrent sur tous les types de variables précédemment définis, par contre les autres opérateurs ne peuvent pas s'appliquer sur des opérandes de type booléen.
2. Le résultat d'une comparaison est un booléen.

2.8.4 Niveaux de priorité

C'est l'ordre dans lequel sont appliquées les différentes opérations d'une expression, par exemple l'expression : x+y*z est évaluée en calculant (y*z) puis le (résultat + x), parce que la multiplication est prioritaire par rapport à l'addition.
Voici la table des priorités classées par ordre décroissant, les opérateurs sur une même ligne ayant une priorité égale.

Niveau	Opération
1	() les parenthèses, fonction()
2	+ - not (unaire)
3	* / div mod and
4	+ - or
5	= <> < > >= <=

Remarque les opérations du même niveau sont appliquées de gauche à droite.

Exemples

1. 10+(4*4) div 2 = 10 + 16 div 2 = 10 + 8 = 18
2. 17 mod (5 div 2)= 17 mod 2 = 1

3. 14 mod 3 + 4.2 / 2 = 2 + 2.1 = 4.1

4. Not(12 <> (3*15/5))and True =
   Not(12 <> 9) and True =
   Not(True) and True =
   False and True =
   False

Autres exemples d'expressions numériques (soit les variables: A=6, B=2, C=3)

- A+B/C = A+(B/C)= 6+(2/3)= 6+0.66=6.66
- A/B mod C = (A/B) mod C= (6/2)mod 3=3 mod 3=0
- B*A + 2*C = (B*A)+(2*C) = (2*6)+(2*3)=12+6=18

Exemples d'expressions incorrectes :

- L'expression booléenne: a < b and c ≤ d, écrite sans parenthèses est incorrecte, car dans la table de priorités, l'opérateur and a une priorité plus élevée que les opérateurs < et ≤, et donc l'expression sera évaluée de la manière suivante:
  a < (b and c) ≤ d , ceci n'a pas de sens. L'expression correcte est:
  (a < b) and (c ≤ d) .
- Soit A, B, et C trois variables réelles, l'expression A/BC est incorrecte, car A/BC = valeur de A sur valeur de la variable de nom BC et non A sur B*C

2.8.5 Les fonctions standards

Fonctions arithmétiques

Fonction	Signification	Type de l'argument	Type du résultat
ABS(x)	$\|X\|$	Entier ou Réel	De même type
SQR(x)	X^2	Entier ou Réel	De même type
SQRT(X)	$\sqrt{X}$	Entier ou Réel ≥ 0	Réel
SIN(X)	Sin(X)	Entier ou Réel	Réel
COS(X)	Cos(X)	Entier ou Réel	Réel
ARTAN(X)	Arctg(X)	Entier ou Réel	Réel
LN(X)	Log(X)	Entier ou Réel	Réel
Exp(X)	e^x	Entier ou Réel ≥ 0	Réel

Fonctions de conversion

Fonction	Signification	Type de l'argument	Type du résultat
Trunc(X)	Partie entière de X	Réel	Entier
Round(X)	Arrondi de X à l'entier le plus proche	Réel	Entier
ORD(X	Rang de X	Entier, Caractère, ou Booléen	Entier
CHR(X)	Caractère ayant le rang X	Entier	Caractère

Fonctions d'ordre

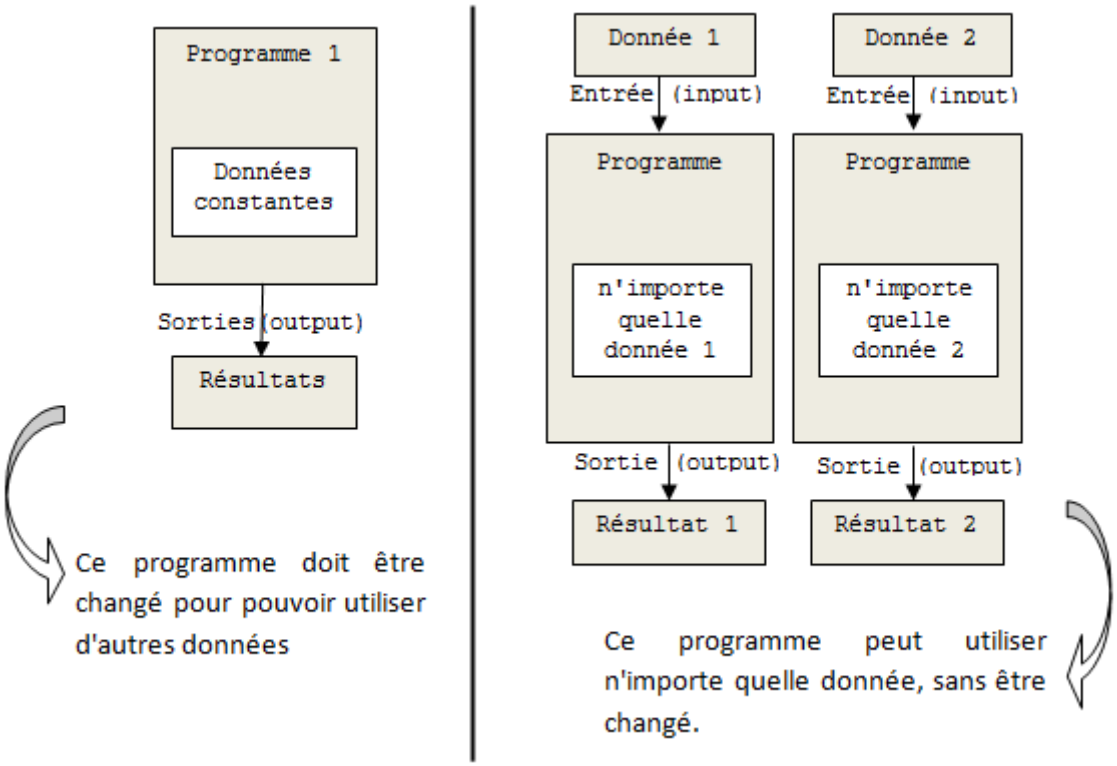
Fonction	Signification	Type de l'argument	Type du résultat
PRED(X)	Valeur qui précède X	Entier, Caractère, ou Booléen	De même type
SUCC(X)	Valeur qui suit X	Entier, Caractère, ou Booléen	De même type
ODD(X)	Vrai si X est impaire, Faux si X est pair	Entier	Booléen

2.9 Les entrées/sorties

2.9.1 Instruction READ/READLN (lire)

Un programme a besoin de données sur lesquelles il opère. Si nous écrivons toutes les valeurs des données dans le programme lui-même, sous forme de constantes ou d'affectation, nous devons réécrire le programme chaque fois que nous voulions l'appliquer à un autre ensemble de valeurs différent. Pour éviter cela, les données doivent être séparées du programme jusqu'à ce qu'il soit exécuté, ensuite les instructions du programme copient les valeurs des données dans les variable du programme. Après la mémorisation de ces valeurs, le programme peut effectuer des calculs avec eux.(voir la figure qui suit)

Le processus de placement des valeurs d'un ensemble de données externes dans des variables d'un programme est appelé : les entrée (input).
Pour entrer des données dans un programme en cours d'exécution, le langage pascal nous offre les procédures READ et READLN.



Read

Cette procédure permet de lire n valeurs entrées par le clavier et les affectent aux variables val1, val2,..,valn, spécifiées comme paramètres.

Syntaxe

Algorithmique	Langage PASCAL
`Lire(val1,val2,...,valn)`	`Read(val1, val2,.., valn);`

- Les valeurs lues peuvent être du type : integer, real, char.
- Il est impossible de lire une valeur booléenne.
- La liste des paramètres d'une instruction read(readln) ne peut contenir que des variables.
- S'il ya plusieurs variables dans une liste de paramètres, les valeurs respectives de ces variables, lorsqu'elles sont entrées par le clavier, doivent être séparées l'une de l'autre par au moins un espace.

Exemple :

*Instructions**	Données	Contenus des variable après Read
1.Read(x);	5	x=5
2.Read(a,b,c);	6 7 8	a=6, b=7, c=8
3.Read(d,e);	28	
	47	d=28, e=47
4.Read(y,z);	46 36.5 8	y=46, z=36.5

*a, b, c, x, y sont de type integer, et z de type real.

- Si le nombre de valeurs entrées est supérieur à celui des paramètres dans l'instruction read, le système conserve le reste des valeurs pour un prochain read s'il existe, si non le système les néglige(voir 4 dans l'exemple précédent).
- S'il n'existe pas suffisamment de valeurs entrées sur une ligne pour remplir les paramètres de read, le système lit automatiquement à partir de ligne d'entrée suivante.(voir 3 dans l'exemple précédent).
- Les deux écritures sont identiques:

```
1   Read(val1, val2,.., valn);
```

et

```
1   READ(val1);
2   READ(val2);
3   ...
4   READ(valn);
```

Readln

READLN permet également de lire des données. La seule différence qui existe entre les deux procédures, c'est que READLN permet un retour à la ligne après la lecture de la liste des variables spécifiées comme paramètres. Ce changement de ligne ne prend effet qu'à la prochaine lecture par un autre READ ou READLN.

Exemple :
Supposons qu'il existe deux valeurs entières sur chaque ligne d'entrée:
10 20
15 16
22 21

*Instructions**	Contenus des variables après Readln
1.Readln(a); Readln(b); Readln(c);	A=10 B=15 C=22
2.Readln(a,b,c); Readln(d,e);	A=10 b=20 c=15 D=22 e=21
3.Readln(a,b); Readln(c,d); Readln(e);	A=10 b=20 C=15 d=16 E=22
4.Readln(a); Readln(b); Readln(c,d,e);	A=10 B=15 C=22, d=21, vous devez entrer une valeur pour la variable e.
5.Readln(a,b); Readln; Readln(c);	A=10 b=20 C=22, Readln sans paramètres provoque un saut de ligne d'entrée.

*a,b,c,d,e sont des variables de type integer.

2.9.2 Instruction WRITE/ WRITELN (écrire)

Write : permet d'afficher tous les paramètres var1,var2,. . . ,varn spécifiés et laisse le curseur à la fin du contenu affiché.

Syntaxe

Algorithmique

```
ecrire(var1,var2,...,varn)
```

Langage PASCAL

```
write(var1, var2,.., varn)
```

- La liste des paramètres d'une instruction write (writeln) peut contenir des constantes, des variables et des expressions,
- L'instruction writeln(x) permet de d'afficher la valeur de x avec un retour à la ligne à la fin de l'affichage.
- L'instruction Writeln sans paramètre permet de sauter une ligne.

Exemple :

```
Program affichage;
Uses wincrt;
var
x, y : integer;
Begin
 writeln('Entrer deux entiers:');
                          { Paramètre est une constante de type string.}
 read(x, y);
```

```
 9    write ('la somme de ') { Paramètre est une constante de type string.}
10    write (x) { Paramètre est une variable.}
11    write (' + ') { Paramètre est une constante de type string.}
12    write (y) { Paramètre est une variable.}
13    write (' = ') { Paramètre est une constante de type string.}
14    write (x+y) { Paramètre est une expression.}
15   end.
```

2.10 L'instruction d'affectation

On appelle AFFECTATION la mise d'une valeur dans une variable. Celle-ci peut être sous forme directe (A := 2, ou A:=B) ou sous forme d'une expression (A:=B*C). Elle est représentée par le signe← dans le langage algorithmique et (:=) dans le langage PASCAL.

2.10.1 Affectation par une variable ou une valeur constante

- L'affectation (variable := valeur) permet de changer la valeur d'une variable.
- L'affectation modifie le contenu de la variable. La valeur de la variable à gauche du signe := est remplacée par la valeur à droite de :=.
- L'affectation (variable1 := variable2) permet de changer la valeur de variable1 par celle de variable2.

2.10.2 Affectation par une expression

L'affectation (variable := expression) est effectuée par :

1. évaluation de l'expression
2. placement du résultat dans la variable à gauche.

Remarques:

- Après une affectation, l'ancienne valeur de la variable est écrasée par la nouvelle valeur.
- Dans une instruction d'affectation, on extrait les valeurs des variables qui se situent à droite du (:=). La partie gauche de (:=) toujours est un nom d'une variable (un identificateur) qui va recevoir le résultat.
  Exemple:
  x:= x + N a pour effet de mettre le résultat de la somme de la valeur de x avec la valeur de N dans la variable x.
- Une écriture du type B*C:=A est IMPOSSIBLE.
- l'écriture correcte est A:=B*C.
  Le signe := signifie "mettre la VALEUR à droite du := dans la zone mémoire désignée à gauche" (mettre le résultat du calcul (contenu de B) fois (contenu de C) dans A).

- Ne pas confondre := avec le = réservé à la comparaison des variables dans un test.
- Une affectation ne peut se faire qu'entre une variable et une expression de même type (si A est réel, impossible de faire A:='bonjour').
- A:=B est différente de B:=A, dans la première instruction c'est la valeur de A qui est modifiée, par contre dans la deuxième, c'est la valeur de B qui est modifiée.
- On ne peut pas écrire variable1:= variable2:= Expression, ni Expression1:= Expression2;
- Si x est un réel et y est un entier: on peut écrire x:=y; mais on ne peut pas écrire y:=x;

Exemple

```
1  Var
2    x,y,z:integer
3   ....
4  X :=2 ;
5  Y :=3 ;
6  Z :=((x+y)*2) ;
7  X:=Z-2;
```

Après exécution : z prend la valeur : 10 et X prend la valeur: 8.

2.11 Exercices d'application

Exercice 1

Ces expressions sont-elles correctes? si oui, donner le type et le résultat d'évaluation.

1. (2.6 + 1) > (4 / 3)
2. (12> 3) > 4
3. Not(12 <> (3*16.8/4))and True
4. 59 < (8*3)
5. (12>24)+(2+4=6)
6. 11 mod 3 + 5.2 / 2
7. False or not false and true

Exercice 2

Ecrire un programme Pascal qui permet de calculer le périmètre d'un cercle à partir de son rayon.

Exercice 3

Ecrire un programme Pascal qui permet de permuter les valeurs de deux variables entières

Solutions

Solution 1

1. (2.6 + 1) > (4 / 3)

   (réel + entier) > (entier/entier)

   3.6(réel) > 1.333 (réel)

   True(booléen) − > Expression correcte

2. (12> 3) > 4

   True(booléen) > 4(entier) − > Expression incorrecte

3. Not(12 <> (3*16.8/4))and True

   Not(true) and true = false and true

   False (booléen) − > Expression correcte

4. 59 < (8*3)

   95 <24

   False(booléen) − > Expression correcte

5. (12>24)+(2+4=6)

   False + true

   erreur − >Expression incorrecte

6. 11 mod 3 + 5.2 / 2

   2 + 2.6 = 4.6 (réel) − > Expression correcte

7. False or not false and true

   False or True and True

False or True

True (booléen) − >Expression correcte

Solution 2

```
1   Program perimetre_cercle;
2   Uses Wincrt;
3   Const
4     PI=3.14159;
5   Var
6     rayon, perimetre : real;
7   Begin
8     Read(rayon);
9     perimetre := 2 * PI * rayon;
10    write('Le perimetre est : ', perimetre);
11  End.
```

Solution 3

Première solution: en utilisant 2 variables

```
1   Program permut;
2   Uses Wincrt;
3   Var
4     a, b : integer;
5   Begin
6     Read(a);
7     Read(b);  { ou bien on écrit directement Read(a, b);
8     a : = a+b;
9     b := a-b;
10    a:= a-b;
11    Writeln('a=',a );
12    Write('b=',b );
13  End.
```

Deuxième solution: en utilisant 3 variables

```
1   Program permut;
2   Uses Wincrt;
3   Var
4     a, b, tmp : integer;
5   Begin
6     Read(a);
7     Read(b);
8     tmp := a;
9     a : = b
```

```
10      b := tmp
11      Writeln('a=',a );
12      Write('b=',b );
13    End.
```

Chapitre 3

Structures de contrôle

Sommaire

3.1 Introduction

Jusqu'ici, les programmes écrits contiennent des suites d'instructions séparées par des points virgules, et qui sont exécutées séquentiellement, c.-à-d. l'une après l'autre, depuis le "Begin" jusqu'à le "end." final.
On peut ne vouloir exécuter certaines instructions que dans des cas bien précis,ou bien vouloir répéter un même bloc d'instructions sur des objets différents, pour se faire, on a recours à des tests ainsi qu'à des instructions conditionnelles, ou des boucles.

Un test est une expréssion booléenne dont le résultat d'évaluation est vrai, ou faux. On peut combiner plusieurs tests en un seul test en utilisant les opérateurs logiques vus au chapitre précédent.
Un bloc d'instructions est un enchaînement d'instructions regroupées et délimitées par un début et une fin (Begin et end).

3.2 Instructions conditionnelles

On distingue deux types d'instructions conditionnelles :

- L'instruction conditionnelle simple : SI...alors... [Ne vouloir exécuter certaines instructions que dans des cas bien précis]
- L'instruction conditionnelles alternative: SI... alors...sinon... [vouloir exécuter différentes instructions selon le résultat du test]

3.2.1 L'instruction conditionnelle simple

Syntaxe

Algorithmique

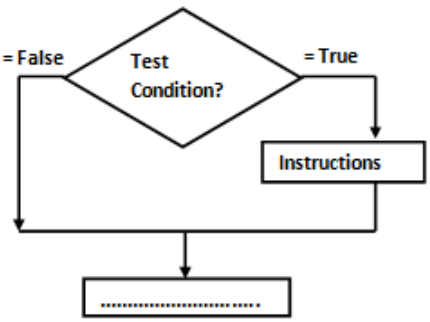
```
1   SI condition ALORS
2      Action(s)
3   FINSI
```

Langage PASCAL

```
1   IF condition THEN
2   BEGIN
3     Instructions ;
4   END;
```

Exécution

- La condition est tout d'abord évaluée,
- les instructions(ou les actions) ne sont exécutées que si la valeur de la condition est égale à True.



Exemple

```
1   Program exemple;
2   Uses wincrt;
3   Const
4     P=10;
5   Var
6     A,b,c :integer ;
7   Begin
8     Readln(a,b,c);
9     If (a>b) and (a<c) then
10    Begin
11      A:= b*2;
12      B:= c-3*p;
13      C:=(c+1)div 3;
14    End;
15  End.
```

-	1ier cas: condition = false [(2>14) and (2>-2)= false]	
	Valeur avant exécution	valeur après exécution
p	10	10
a	2	2
b	14	14
c	-2	-2

-	2ième cas: condition = true [(12>4) and (12>8)=true]	
	Valeur avant exécution	valeur après exécution
p	10	10
a	12	8
b	4	-22
c	8	3

3.2.2 L'instruction conditionnelle alternative

Syntaxe

Algorithmique

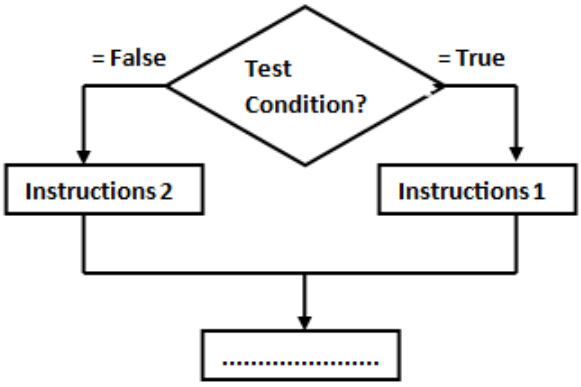
```
1   SI condition ALORS
2      Action(s)1
3   SINON
4      Action(s)2 ;
5   FINSI
```

Langage PASCAL

```
1   IF condition THEN
2   BEGIN
3     Instructions 1;
4   END
5   Else
6   Begin
7     Instructions2;
8   End;
```

Exécution

- La condition est tout d'abord évaluée,
- Si le résultat du calcul égale à true, le bloc d'instructions 1 est exécuté,
- Si le résultat du calcul égale à false, le bloc d'instructions 2 est exécuté.



Exemple

```
1   Program exemple;
2   Uses wincrt;
3   Var
4     A,b,c,d :integer ;
5   Begin
6     Readln(a,b,c);
7     If (a>b) or (c<d) then
8     Begin
9       A:= b*3;
10      B:= (c-b)*2;
11    End
12    Else
13    Begin
14      B:=a*3 mod 2;
15      C:=d div 2 - 2;
16    End;
17  End.
```

-	1ier cas: condition = false [(8>12) or (6>-2)= false]	
	Valeur avant exécution	valeur après exécution
a	8	8
b	12	0
c	6	-3
d	-2	-2

-	2ième cas: condition = true [(10>8) or (9>4)=true]	
	Valeur avant exécution	valeur après exécution
a	10	24
b	8	2
c	9	9
c	4	4

Remarques

- Ne jamais mettre un point virgule ";" avant le else;
- le 'Begin end' est inutile, s'il n'y a qu'une seule instruction après le Then ou apès le Else.
- On n'a le droit qu'à deux options selon le résultat du test: un seul then et un seul else. Pour traiter d'autres cas, on doit combiner(imbriquer) plusieurs instructions conditionnelles.

Exemple

```
1   program IF_imbriques;
2   var
3   a : integer;
4   begin
5   a := 4;
6   if (a = 1)
7     then  writeln('Valeur de a= 1' )
8     else if ( a = 2 ) then  writeln('Valeur de a= 2 )
9                   else if( a = 3)then writeln('valeur de a= 3' )
10                                  else
11                                   begin
12                                     writeln('a est différente de 1,2,et 3');
```

```
13                                    writeln(' a=  ', a );
14                                  end;
15  end.
```

Après exécution du programme, le système affiche:

a est différente de 1,2, et 3
a= 4

- On peut changer une instruction conditionnelle alternative par deux instructions conditionnelles simples comme suit :

```
1  If condition then Instruction1
2                else Instruction2;
```

devient

```
1  If condition then Instruction1;
2  If Not(condition) then Instruction2;
```

Attention: ce n'est pas toujours correct

Exemple

If-then-else

```
1  x:=-10;
2  If x<0 then x:=x+10
3         else x:=x+20;
4  write('x= ', x);
```

If-then If-then

```
1  x:=-10;
2  If x<0 then x:=x+10;
3  If x>=0 then x:=x+20;
4  write('x= ', x);
```

Le résultat n'est pas le même, dans le premier cas x= 0, mais dans le deuxième cas x= 20.

3.3 Instructions répétitives

3.3.1 Instruction For - do (boucle For - do)

Cette instruction permet de répéter une séquence d'instructions un nombre fixe de fois.

Syntaxe

Algorithmique

```
1   Pour(indice <- inf à sup)faire
2   Action(s) ;
3   FINPOUR
```

Langage PASCAL

(1)

```
1   For indice:= inf to sup do
2   Begin
3      Bloc d'instructions;
4   End;
```

Ou bien (2)

```
1   For indice:= sup downto inf do
2   Begin
3      Bloc d'instructions;
4   End;
```

- Indice est une variable entière, souvent appelée compteur de la boucle.
- inf et sup sont, respectivement, les bornes inférieures et supérieures de la boucle. Ceux sont deux constantes entières ou deux variables entières initialisées.
- Le bloc d'instructions est répété pour toutes les valeurs de l'indice comprise (inclusivement) entre inf et sup.
- La variable indice est augmentée automatiquement d'une unité à chaque passage dans la boucle.
- Cette forme de boucle est utilisée si on connaît d'avance le nombre de fois à répéter les instructions.
- Le bloc d'instructions est exécuté (val-sup - val-inf + 1) fois.
- Si on veut aller dans un ordre décroissant (de sup jusqu'à inf), on remplace TO par DOWNTO (voir syntaxe (2)).
- La variable indice peut être utilisée dans le bloc d'instructions, mais pas modifiée
- Si le bloc d'instructions ne contient qu'une seule instruction, le "Begin end" devient inutile.

Exemple 1

```
1   Program exemple_1;
2   Uses wincrt;
3   Var
4     puis,i, x :integer ;
5   Begin
6           puis:=1; read(x);
7           For i:=1 to 4 do
```

```
8           puis:= puis*x;
9           Writeln('x à la puissance 4 = ', puis);
10  End.
```

* Si x =3 par exemple, l'instruction puis := puis*x est exécutée (4-1+1 fois c.-à-d. 4 fois)

Le résultat est:3^4=81.

puis	i	Numéro d'itération
3	1	1
9	2	2
27	3	3
81	4	4

Exemple 2

```
1   Program exemple_2;
2   Uses wincrt;
3   Var
4     som,i, n :integer ;
5   Begin
6           read(n); som:=0;
7           For i:=1 to n do
8           som:= som+i;
9           Writeln('La somme des entier compris entre 1 et n est: ', som);
10  End.
```

* Si n =4 par exemple, l'instruction som := som+i est exécutée (4-1+1 fois c.-à-d. 4 fois)

Le résultat est:1+2+3+4=10.

som	i	Numéro d'itération
0+1=1	1	1
1+2=3	2	2
3+3=6	3	3
6+4=10	4	4

- Comme on peut le voir dans cet exemple, on peut utiliser la valeur de indice dans le bloc d'instructions.

3.3.2 Instruction While - do (boucle While - do)

Cette instruction permet de répéter une séquence d'instructions tant que une condition est vraie.

Syntaxe

Algorithmique

```
1   TANTQUE condition FAIRE
2   Action(s) ;
3   FINTANTQUE
```

Langage PASCAL

```
1   WHILE condition DO
2   BEGIN
3     Instruction(s) ;
4   END;
```

- Condition est une expression booléenne
- Le bloc d'instructions est répété, tant que la condition est vraie.
- Si la condition est fausse dès le début, le bloc d'instructions n'est jamais exécuté.
- Cette forme de boucle est utilisée si au moment où on écrit la boucle, on ne connaît pas le nombre de fois à répéter le bloc d'instructions.
- Les instructions d'une boucle while continuent de s'exécuter tant que la condition n'est pas fausse. Pour éviter une boucle infinie, il faut impérativement ajouter une instruction qui rend la condition fausse à un moment donné.
- Aussi, Si le bloc d'instructions ne contient qu'une seule instruction, le "Begin end" devient inutile.

Exemple

```
1   Program exemple;
2   Uses wincrt;
3   Var
4     i, Som, nbre :integer ;
5   Begin
6   Writeln('Donnez un nombre');
7   Som:=0;
8   i:=1;
9   read(nbre);
10  While i <= nbre do
11  begin
12    Som :=som+i ;
13    i:=i+1;
14  End ;
15  Writeln('somme obtenue=', somme) ;
16  End.
```

Voici la trace d'exécution si nbre = 5:

Condition i<=5	Instructions: 1. i<=nbre 2. som:= som+i; 3. i:=i+1;	i	som	numéro d'itération
True	(1)-(2)-(3)	1	1	1
True	(1)-(2)-(3)	2	3	2
True	(1)-(2)-(3)	3	6	3
True	(1)-(2)-(3)	4	10	4
True	(1)-(2)-(3)	5	15	5
False	(1)	6	15	fin

Remarque:

L'instruction i:=i+1 permet de modifier la valeur de la variable i qui compose la condition (i<=nbre), si on oublie cette instruction, le programme s'exécutera infiniment

3.3.3 Instruction Repeat-until (boucle Repeat-until)

Cette instruction permet de répéter une séquence d'instructions tant que une condition est fausse.

Syntaxe

Algorithmique

```
1   REPETER
2   Action(s) ;
3   JUSQU'A Condition
```

Langage PASCAL

```
1   REPEAT
2        Instruction(s);
3   UNTIL Condition;
```

- Les instructions de la boucle sont répétées jusqu'à ce que la valeur de la condition égale à vrai.
- Même si la condition est vraie dès le début, les instructions sont au moins exécutées une fois.

- Pas besoin d'encadrer les instructions par un "begin end", le "repeat until" joue déjà ce rôle.

Exemple

```
1   PROGRAM multiples ;
2   USES wincrt ;
3   VAR n :integer ;
4   Begin
5           n :=100 ;
6           REPEAT
7           n :=n+1 ;
8           IF (n mod 9 =0) THEN Writeln(n) ;
9           UNTIL n>200 ;
10  End.
```

Ce programme affiche à l'écran tous les multiples de 9 compris entre 100 et 200.
Pour chaque nombre n compris entre 100 et 200, on calcule le reste de sa division par 9, si ce reste est égale à 0, alors n est un multiple de 9.

3.3.4 Différence entre les deux boucles conditionnelles

While-do

- Le test est effectué avant d'entrer dans la boucle.
- On sort de la boucle si la condition est fausse
- Les instructions de la boucle peuvent ne jamais être exécutées, si dès la première fois la condition est fausse.

Repeat-until

- Le test est effectué après les instructions de la boucle
- On sort de la boucle si la condition est vraie
- Les instructions de la boucle sont exécutées au moins une fois.

ATTENTION:

Pas de point virgule ";" après le do dans une boucle, et avant le else dans une instruction conditionnelle.

3.3.5 Les boucles imbriquées

Les trois types de boucles peuvent être imbriquées; c.-à-d. le bloc d'instructions d'une boucle contient une autre boucle. Les boucles imbriquées sont surtout utilisées pour la manipulation des matrices

Exemple:

Ecrire un programme Pascal qui permet de calculer et d'afficher la somme suivante:

$$S = 1^5 + 2^5 + 3^5 + ... + 10^5$$

```
1   Program Exemple_boucles_imbriquees;
2   Uses wincrt;
3   var
4     s, i, j, puis :integer;
5   Begin
6    s:=0;
7    For i:=1 to 10 do {boucle 1}
8    Begin
9     puis:=1;
10    for j:=1 to 5 do {boucle 2}
11     puis:=puis*i;
12    s:= s+ puis;
13   end;
14   write('S= ', s);
15   end.
```

3.3.6 Passer d'une boucle à une autre

1. while-do —> repeat-until

Devient

```
1   While condition do
2   begin
3    Instructions
4   end;
```

```
1   if condition then
2   repeat
3    Instructions
4   until Not(condition);
```

2. For-do —> while-do

Devient

```
1   For indice:= inf to sup do
2   begin
3    Instructions
4   end;
```

```
1   indice:= inf;
2   while indice <= sup do
3   Begin
4    Instructions;
5    indice:= indice+1;
6   end;
```

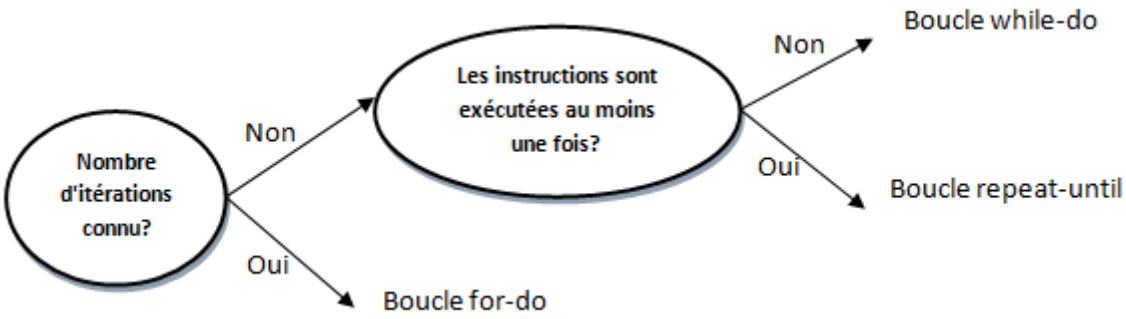
3. For-do —> Repeat-until

Devient

```
1  For indice:= inf to sup do
2  begin
3   Instructions
4  end;
```

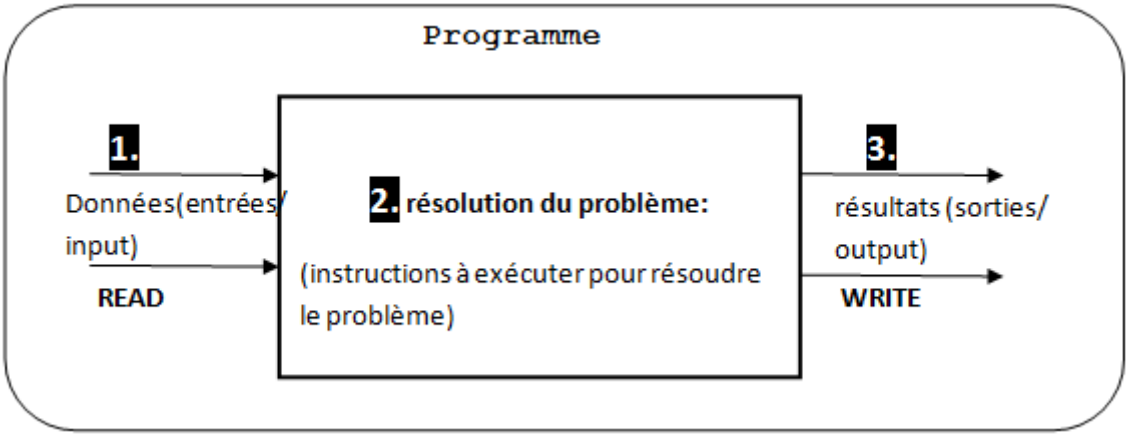
```
1  indice:= inf;
2  if indice <= sup then
3  Repeat
4   Instructions;
5   indice:= indice+1;
6  Until indice > sup;
```

3.3.7 Choisir la boucle for-do, while-do ou repeat-until



3.3.8 Démarche à suivre pour écrire un programme

Pour construire un programme, on doit suivre les étapes suivantes:



1. Introduire les données: faire entrer les valeurs des données.
2. Résoudre le problème: manipuler les données pour obtenir la solution au problème donné, cette étape se diffère selon le problème posé.

3. Affichage du résultat obtenu.

3.4 Exercices d'application

Exercice 1

Ecrire un programme Pascal qui permet de tester si un nombre entier donné par l'utilisateur est pair ou non.

Exercice 2

Ecrire un programme Pascal qui permet d'afficher le minimum de trois valeurs réelles distinctes données par l'utilisateur.

Exercice 3

Ecrire un programme pascal qui calcule et affiche a^b, telque a et b sont deux entiers donnés par l'utilisateur.

Exercice 4

Ecrire un programme pascal qui calcule et affiche la somme suivante: $1/2 + 1/4 + 1/6 + ... + 1/100$, en utilisant la boucle For-do puis la boucle While-do.

Exercice 5

Ecrire un programme pascal qui permet de demander un mot de passe à l'utilisateur, puis vérifier et afficher si le mot de passe est correct ou non

Solutions

Solution 1

```
1  PROGRAM pair_impair;
2  USES wincrt ;
3  VAR
4    n :integer ;
5  Begin
6    Read(n);
7    If (n mod 2 <> 0) then write(n, ' est impair');
8    If (n mod 2 = 0) then write(n, ' est pair');
9  End.
```

ou bien

```
1  PROGRAM pair_impair;
2  USES wincrt ;
3  VAR
4    n :integer ;
5  Begin
```

```
6     Read(n);
7     If (n mod 2 <> 0) then write(n, ' est impair')
8                             else write(n, ' est pair');
9   End.
```

Solution 2

```
1   PROGRAM minimum;
2   USES wincrt ;
3   VAR
4     a,b,c :integer ;
5   Begin
6     Read(a,b,c);
7     If (a < b)and (a < c) then write('Le minimum est: ', a);
8     If (b < a)and (b < c) then write('Le minimum est: ', b);
9     If (c < a)and (c < b) then write('Le minimum est: ', c);
10  End.
```

ou bien

```
1   PROGRAM minimum;
2   USES wincrt ;
3   VAR
4     a,b,c :integer ;
5   Begin
6     Read(a,b,c);
7     If (a <b)then if (a < c) then write('Le minimum est: ', a);
8                             else write('Le minimum est: ', c);
9                   else if (b < c) then write('Le minimum est: ', b);
10                            else write('Le minimum est: ', c);
11  End.
```

Solution 3

```
1   Program Puissance;
2   Uses wincrt;
3   Var
4           a, b, i, puis : integer;
5   Begin
6   Read(a,b);
7   puis:=1;
8   For i:=1 to b do
9   puis:= puis*a;
10  Write(a,  'puissance ', b '=' , puis);
11  End.
```

Solution 4

1. Boucle For-do

```
1   Program Somme;
2   Uses wincrt;
3   Var
4           i, som : integer;
5   Begin
6   som:=0;
7   For i:=2 to 100 do
8   if i mod 2 = 0 then som:= som+1/i;
9   Write('la somme= ', som);
10  End.
```

2. Boucle While-do

```
1   Program Somme;
2   Uses wincrt;
3   Var
4           i, som : integer;
5   Begin
6   som:=0;
7   i:=2;
8   While (i<= 100) do
9   Begin
10  som:= som+1/i;
11  i:=i+2;
12  end;
13  Write('la somme= ', som);
14  End.
```

Solution 5

```
1   Program Mot_de_passe;
2   Uses wincrt;
3   Const
4     True_Password = 'password';
5   Var
6     Password_Correct : Boolean;
7     Word : String;
8   Begin
9     Password_Correct := False;
10    Repeat
11        Write('Entrez le code secret : ');
12        Readln(Word);
13        if Word = True_Password then Password_Correct := True;
14     until (Password_Correct := True) ;
15  End.
```

Chapitre 4

Les structures de données complexes: Les tableaux

Sommaire

4.1 Introduction

Jusqu'ici, nous avons utilisé des types de variables dont les valeurs sont simples(entière, réelle, caractère, ...). Lorsqu'on doit manipuler un grand nombre de valeurs, ce type de variables devient inefficace. L'idée consiste donc, lorsque les données sont du même type, à les ranger dans une structure de données linéaire appelée tableau, ou array en anglais.

4.2 Tableau à une dimension (vecteur)

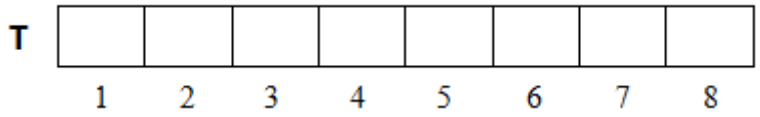
4.2.1 Définition

Un tableau à une dimension t est une collection finie de cases (mémoires) désignées par des indices, contenant des valeurs, de même type et qui peuvent être manipulées individuellement.

Un tableau à une dimension (ou vecteur) est caractérisé par:

- Son nom (identificateur du vecteur)
- Sa dimension
- Le type de ses éléments.

Schématiquement, un vecteur T contenant 8 éléments peut être représenté par :



T: est le nom du vecteur,
T[1] : désigne le contenu de l'élément numéro 1 du vecteur T,
T[2] : désigne le contenu de l'élément numéro 2 du vecteur T, ...
T[i] : désigne le contenu de l'élément numéro i du vecteur T (i est une variable scalaire).

Exemples

12	14	1	3	10

Un vecteur de 5 valeurs entières

'A'	'K'	'Z'	'Y'	'F'

Un vecteur de 5 valeurs caractères

4.2.2 Syntaxe de déclaration

La syntaxe de déclaration est:

- En algorithmique:

```
1   Nom_tableau : tableau[1..dimension] type_elements
```

- En PASCAL:

```
1   Nom_tableau :array[1..dimension]of type_elements ;
```

Exemples:

1. Déclarer le type puis une variable de ce type:

```
1   type
2      tab=array[1..10]of integer;
3   var
4      t1, t2: tab;
```

2. Déclarer le tableau:

```
1   var
2      t1: array[1..10] of integer;
```

4.2.3 Opérations sur les vecteurs (T: array[1..10] of integer)

1. Initialiser la 5ième case de T à 0 :

```
1   T[5]:=0 ;
```

2. Initialiser la 8ième case de T à une valeur entrée par l'utilisateur :

```
1   Readln(T[8]);( ou bien read(T[8]);)
```

3. Initialiser tout le vecteur à 1:

```
1   For  i:=1 to 10 do
2           T[i]:=1;
```

4. Initialiser tout le vecteur à des valeurs entrées par l'utilisateur (lecture du vecteur):

```
1   For i:= 1 to 10 do
2           Readln(T[i]);
```

5. Afficher les éléments d'un vecteur:

   (a) Afficher la valeur de la case numéro 6:

   ```
   1   Write(T[6]);
   ```

   (b) Affichage des éléments d'un vecteur (affichage horizontal: sur la même ligne):

   ```
   1   For i:=1 to 10 do
   2           Write(T[i], '   ');
   ```

(c) Affichage des éléments d'un vecteur (affichage vertical: chaque élément sur une ligne):

```
1   For i:=1 to 10 do
2           Writeln(T[i]);
```

Remarques:
La seule opération globale possible sur les vecteurs est l'affectation. Si R et A sont deux vecteurs de même dimension et type de contenu, l'instruction R:=A; remplace le contenu du vecteur R par celui du vecteur A. Toutes les autres opérations portant sur les vecteurs se font case par case, donc en utilisant des boucles.

Exemple:

```
1   var
2     R,A: array[1..10] of real;
3     T: array[1..5] of integer;
4     CH: array[1..5] of char;
```

Les deux écritures R:=A et A:= R sont correctes mais R:= CH, A:= CH, CH:=A, T:= R, T:=CH et CH:= R sont incorrectes.

4.3 Tableaux à deux dimensions (Matrices)

Dans d'autres applications, on a besoin d'utiliser des tableaux à deux dimensions, qui sont capables de manipuler des structures plus complexes. Par exemple: supposons qu'on veut stocker les notes d'un groupe d'étudiants de différentes sections. Comme il est représenté par le tableau qui suit, chaque élément de ce tableau est lié à une ligne (le numéro d'étudiant) et une colonne (le numéro de la section); ce sont les deux dimensions du tableau. Les données du tableau sont des réels.

N d'étudiant (1ier index)	N de section (2ieme index)				
	1	2	3	4	5
1	17.5	10	12	19.75	15
2	10	11.25	12.5	04.5	13.5
3	03.5	12	15	16	16.25
4	15	07.5	13.75	10.25	09
...	...	...	...	...	...
100	09.75	10	18	03.5	11

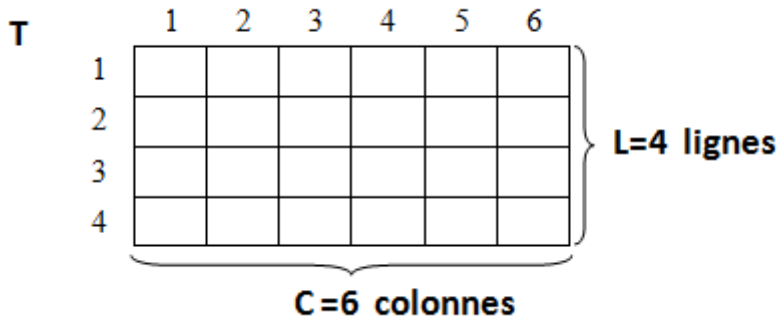
4.3.1 Définition

Un tableau à deux dimensions (ou matrice) peut être vu comme un tableau à une seule dimension L dont chaque élément est un tableau à une seule dimension C, où L est le nombre de lignes et C est le nombre de colonnes. Un tableau à deux dimensions (matrice) contient L*C éléments.

Donc, une matrice est caractérisée par:

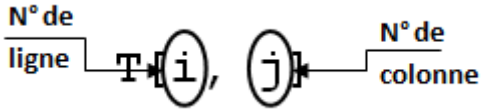
- son nom (identificateur de la matrice)
- ses dimensions (le nombre de lignes et de colonnes)
- le type de ses éléments.

Schématiquement, une matrice T contenant 4 lignes et 6 colonnes peut être représentée par :



T: est le nom de la matrice,
T[1,1] : désigne le contenu de l'élément situé à l'intersection de la ligne 1 et la colonne 1,
T[2,3] : désigne le contenu de l'élément situé à l'intersection de ligne 2 et la colonne 3 de la matrice T,
...
T[i,j] : désigne le contenu de l'élément situé à l'intersection de ligne i et la colonne j de la matrice T (i, j sont des variables entières).

Attention Toujours le premier indice est le numéro de ligne et le second est le numéro de colonne.



4.3.2 Syntaxe de déclaration

La syntaxe de déclaration est:

- En algorithmique:

```
1   Nom_tableau : tableau[1.. nb_lignes, 1..nb_colonnes] type_elements
```

- En PASCAL:

```
1   Nom_tableau :array[1.. nb_lignes, 1..nb_colonnes]of type_elements ;
```

Exemple

```
1   var
2      t1: array[1..10, 1..20] of integer;
```

4.3.3 Opérations sur les matrices (MAT: array[1..10,1..20] of integer)

1. Initialiser la case située à la 5ième ligne et la 7ième colonne de MAT à 0 :

```
1   MAT[5,7]:=0 ;
```

2. Initialiser la case située à la 5ième ligne et la 8ième colonne à une valeur entrée par l'utilisateur :

```
1   Readln(MAT[5,8]);( ou bien read(MAT[5,8]);)
```

3. Initialiser toute la matrice à 1:

```
1   For  i:=1 to 10 do
2           For j:=1 to 20 do
3                   MAT[i,j]:=1;
```

4. Initialiser toute la matrice à des valeurs entrées par l'utilisateur (lecture de la matrice):

```
1   For i:= 1 to 10 do
2           For j:=1 to 20 do
3                   Readln(MAT[i,j]);
```

5. Afficher les éléments de la matrice:

   (a) Afficher la valeur de la case située à la 5ième ligne et la 7ième colonne de MAT:

```
1   Write(MAT[5,7]);
```

   (b) Afficher tous les éléments de la matrice:

```
1   For i:=1 to 10 do
2   Begin
3           For j:=1 to 20 do
4           Write(T[i], '   ');
5           Writeln;
6   end;
```

4.4 Exercices d'application

Exercice 1

Ecrire un programme Pascal qui permet de normaliser un vecteur A contenant 20 réels(remplacer les A_i par $\frac{A_i - min_A}{max_A - min_A}$

Exercice 2

Ecrire un programme Pascal qui permet de calculer la somme de deux matrices A et B contenant 4 lignes et 3 colonnes

Exercice 3

Ecrire un programme Pascal qui permet de calculer la transposée d'une matrice A de 2 lignes et 3 colonnes

Exercice 4

Ecrire un programme Pascal qui permet de calculer la somme des éléments de la diagonale d'une matrice carrée d'ordre 3.

Solutions

Solution 1

```
1   Program Normalisation;
2   Uses wincrt;
3   Var
4      A: array[1..20]of real;
5      max, min: real;
6      i:integer;
7   Begin
8   for i:=1 to 20 do
9   Read(A[i]);
10  min:= A[1];
11  max:= A[1]
12  for i:=2 to 20 do
13  Begin
14   if A[i]< min then min:= A[i];
15   if A[i]> max then max:= A[i];
16  end;
17  For i:=1 to 20 do
18  Begin
19  A[i]:= (A[i]- min)/(max-min);
20  Write (A[i],'  ');
21  end;
22  end.
```

Solution 2

```
1   Program Somme_matrices;
2   Uses wincrt;
3   Var
4      A, B, SOM: array[1..4,1..3]of real;
5      i, j:integer;
6   Begin
7   For i:=1 to 4 do
8   For j:=1 to 3 do
9   Begin
10  Read(A[i,j]);
11  Read(B[i,j]);
12  SOM[i,j]:= A[i,j]+B[i,j];
13  end;
14  For i:=1 to 4 do
15  Begin
16  For j:=1 to 3 do
17  Write(SOM[i,j],' ');
18  Writeln;
19  end;
20  end.
```

Solution 3

```
1   Program transposee_matrice;
2   Uses wincrt;
3   Var
4      A, A_T: array[1..2,1..3]of real;
5      i, j:integer;
6   Begin
7   For i:=1 to 2 do
8   For j:=1 to 3 do
9   Begin
10  Read(A[i,j]);
11  A_T[j,i]:= A[i,j];
12  end;
13  For i:=1 to 2 do
14  Begin
15  For j:=1 to 3 do
16  Write(A_T[i,j],' ');
17  Writeln;
18  end;
19  end.
```

Solution 4

```
1   Program Somme_diagonale;
```

```
2   Uses wincrt;
3   Var
4      A: array[1..3,1..3]of integer;
5      i, j, som :integer;
6   Begin
7   For i:=1 to 3 do
8   For j:=1 to 3 do
9   Read(A[i,j]);
10  For i:=1 to 3 do
11  som:= som+ A[i,i];
12  Write('la somme des éléments de la diagonale = ', som);
13  end.
```

Chapitre 5

Les structures de données complexes: Les chaines de caractères

Sommaire

5.1 Définition

Une chaîne de caractères est une séquence de caractères (lettres, chiffres ou symboles) de longueur variable au cours de l'exécution et d'une taille prédéfinie entre 1 et 255.
Une chaine de caractères est caractérisée par :

- Une taille maximale fixe,
- Une longueur courante variable, jamais supérieure à la taille.

5.2 Syntaxe de déclaration

En Pascal, les chaînes de caractères forment le type STRING. Elle peuvent être déclarées de deux manières

```
1  Var
2  Nom_variable : string [constante];
```

ou

```
1  Var
2  Nom_variable : string ;
```

Où constante: indique la taille maximale de la chaîne de caractères. Si cette taille n'est pas précisée, une taille par défaut de 255 caractères est appliquée.

- Il est possible de déclarer des constantes de type chaîne de caractères(STRING).
  Exemple:

```
1  Const
2   CHAINE1 = 'program';
3   CHAINE2 = 'mation';
4   VIDE = ''; {chaîne vide}
```

- Les constantes de type chaîne se délimitent par des apostrophes (ex: 'Langage PASCAL').
- Pour mettre une apostrophe dans la variable C, de type string, on peut faire C:="" : la 1ère apostrophe pour dire qu'il va y avoir un caractère, les 2 suivantes qui symbolisent l'apostrophe (car une seule apostrophe veut dire fin de la chaine de caractères), la dernière qui signifie la fin de la chaine de caractères.
- Une chaîne de longueur nulle ne comprend aucun caractère : c'est la chaîne vide.
- Les opérateurs =,<>,<,>, $\leq$, $\geq$ permettent de comparer deux chaînes de caractères, et fournissent un résultat logique en fonction du code interne des caractères (code ASCII).
  Exemple:
  'ABCD' < 'AFGH' est vraie car le code de B est inférieure au code de F.

- La valeur numérique 125 et la suite de caractères '125' sont deux objets totalement différents, et la conversion de l'un à l'autre peut se faire en utilisant des procédures prédéfinies du langage.
- On peut affecter un caractère à une chaîne de caractères, mais pas le contraire.
  Exemple:

```
1  Var
2   ch: string;
3   c: char;
4  Begin
5   c:='a';
6   ch:=c; {affectation correcte}
7   ch:='x';
8   c:=ch; {affectation incorrecte}
9  end.
```

5.3 Opérations sur les chaines de caractères

5.3.1 Lecture/Ecriture d'une chaîne de caractères

La procédure Readln permet de lire une chaîne de caractères et la procédure write(ln) permet de l'afficher.

5.3.2 Accès aux éléments d'une chaîne de caractères

L'accès aux caractères composant une chaîne de caractères se fait par la même notation que celle utilisée pour les éléments d'un tableau. Seulement l'accès aux caractères présents est possible.
Une chaîne de caractères peut être traitée comme un vecteur de caractères, ces caractères sont indéxés à partir de 1, alors:

Le premier caractère a pour indice 1,
le deuxième à pour indice 2,
...
le dernier à pour indice la longueur de la chaîne.

Exemple:

```
1  Var
2    Chaine :string[20] ;
3  ...
4  Chaine := 'ABCD';
```

Chaine[1] vaut 'A'
Chaine[2] vaut 'B'
Chaine[3] vaut 'C'
Chaine[4] vaut 'D'
Chaine[5] n'existe pas.

On peut, aussi, modifier un seul caractère de la chaîne.
Exemple:

```
1  Var
2    Ch :string[20] ;
3  Begin
4   Ch := 'ABCD';
5   ch[2]:='6'; { ch devient 'A6CD'}
6  end.
```

5.4 Les fonctions prédéfinies relatives aux chaînes de caractères

Soient ch, ch1,ch2,...chn,sousch des chaînes de caractères et indice, long des nombre entiers. On a :

- Length(ch) : Renvoie un entier représentant la longueur courante de la chaîne ch.
  Exemple:

```
1  ...
2  L:= length('programmation'); {L=13}
3  ...
```

- Concat(ch1,ch2,...,chn) : Concaténation de plusieurs chaînes chi. La chaîne résultante est la concaténation de toutes les chaînes transmises en paramètres. L'opérateur + donne le même résultat que la fonction Concat.
  Exemple:

```
1  ...
2  ch:=concat('program', 'mation'); {ch='programmation'}
3  ch:=concat('Deux', ' ', 'chaines'); {ch='Deux chaines'}
4  ch:= 'Deux'+ ' '+ 'chaines';{ch='Deux chaines'}
5  ...
```

- Pos(sousch,ch) : Recherche l'occurrence de sousch dans ch, et renvoie une valeur entière représentant la position du premier caractère de sousch dans ch si sousch existe. Si Pos ne trouve pas sousch, elle renvoie zéro.
  Exemple:

```
1  ...
2  p:=pos('bc', 'abcdefabcdegh'); {p=2}
3  p:=pos('BC', 'abcdefabcdegh'); {p=0}
4  ...
```

- Copy(ch,indice,long) : renvoie une copie de la sous chaîne de longueur long commençant au caractère ch[indice].
  Exemple:

```
1   ...
2   chaine:=copy('Programmation', 1, 7); {chaine='Program'}
3   ...
```

5.5 Les procédures prédéfinies relatives aux chaînes de caractères

Soient ch, sousch des chaînes de caractères et indice, nb des nombre entiers. On a :

- Delete (ch, indice, nb) : supprime Nb caractères dans la chaîne ch en commençant à l'emplacement indice. Si indice est supérieure à la longueur de ch, aucun caractère n'est supprimé. Si Nb est supérieure au nombre de caractères restant dans la chaîne ch à partir de indice, seule la fin de la chaîne est effacée.
  Exemple:

```
1   ...
2   Chaine := 'programmation';
3   delete(Chaine, 8,2); {chaine='programtion'}
4   ...
```

- Insert (sousch, ch, indice) : insère Sousch dans ch en position indice. Si la chaîne résultante a plus de 255 caractères, elle est tronquée après le 255ème caractère.
  Exemple:

```
1   ...
2   Chaine := 'programmation';
3   insert('pascal', Chaine, 8); {chaine='programpascalmation'}
4   ...
```

- Str (x, ch) : Convertit le numérique x (entier ou réel) en chaîne de caractères de nom ch.
  Exemple:

```
1   ...
2   str(12,chaine); {chaine='12'}
3   str(12.5, chaine); {chaine='1.2500000E+01'}
4   ...
```

- Val (ch, x, codeerreur) : convertit la chaine de caractères ch en un nombre x et renvoie codeerreur(de type entier) qui est égale à 0 si la conversion est possible.
  Exemple:

```
1   ...
2   val('516.4', x, Codeerreur);
3   ...
```

  1. Si x est déclarée Real: la variable x contient le réel 516.4, et Codeerreur contient 0.
  2. Si x est déclarée integer: la variable x contient 0, Codeerreur contient 4, position du point erroné dans l'écriture d'un entier.

5.6 Exercices d'application

Exercice 1

Ecrire un programme PASCAL qui lit une chaîne de caractère CHAINE, puis affiche les informations suivantes:

1. Le premier caractère de la chaîne.
2. Le dernier caractère de la chaîne.
3. La chaîne de caractère sauf le premier caractère.
4. La chaîne de caractère sauf le dernier caractère.

Exercice 2

Ecrire un programme Pascal qui vérifie si un mot donné par l'utilisateur est un palindrome. Un palindrome est un mot qui se lit de la même façon de gauche à droite ou de droite à gauche (ex. ICI, RADAR)

Solutions

Solution 1

```
1   Program chaine_de_car;
2   Uses wincrt;
3   var
4           chaine, ch_sauf_pr, ch_sauf_dr: string[30];
5           pr_char, de_char: char;
6           l :integer;
7   Begin
8    Writeln('Entrez une chaine:');
9    Readln(chaine);
10   l:=length(chaine);
11   pr_char:=chaine[1];
12   de_char:=chaine[l];
13   ch_sauf_pr:=copy(chaine,2,l-1);
14   ch_sauf_dr:=copy(chaine,1,l-1);
15   writeln('Le premier caractère est: ', pr_char);
16   writeln('Le dernier caractère est: ', de_char);
17   write('La chaine de caractère sauf le premier caractère est: ');
18   writeln(ch_sauf_pr);
19   write('La chaine de caractère sauf le dernier caractère est: ');
20   writeln(ch_sauf_dr);
21  end.
```

Ou bien

```
1   Program chaine_de_car;
2   Uses wincrt;
```

```
3   var
4           chaine: string[30];
5   Begin
6   Writeln('Entrez une chaine:');
7   Readln(chaine);
8   writeln('Le premier caractère est: ', chaine[1]);
9   writeln('Le dernier caractère est: ', chaine[length(chaine)]);
10  write('La chaine de caractère sauf le premier caractère est:  ');
11  writeln(copy(chaine,2,length(chaine)-1));
12  write('La chaine de caractère sauf le dernier caractère est: ');
13  write(copy(chaine,1,length(chaine)-1));
14  end.
```

Ou bien

```
1   Program chaine_de_car;
2   Uses wincrt;
3   var
4           chaine, ch_sauf_pr, ch_sauf_dr: string[30];
5           pr_char, de_char: char;
6           l,  i :integer;
7   Begin
8    Writeln('Entrez une chaine:');
9    Readln(chaine);
10   l:=length(chaine);
11   pr_char:=chaine[1];
12   de_char:=chaine[l];
13   ch_sauf_pr:='';
14   ch_sauf_dr:='';
15   for i:=2 to l do
16      ch_sauf_pr:= ch_sauf_pr + chaine[i];
17         for i:= 1 to (l-1) do
18             ch_sauf_dr:= ch_sauf_dr + chaine[i];
19   writeln('Le premier caractère est: ', pr_char);
20   writeln('Le dernier caractère est: ', de_char);
21   write('La chaine de caractère sauf le premier caractère est: ');
22   writeln(ch_sauf_pr);
23   write('La chaine de caractère sauf le dernier caractère est: ');
24   writeln(ch_sauf_dr);
25  end.
```

Solution 2

```
1   Program Palindrome;
2   Uses Wincrt ;
3   Var
4   ch, ch_inv : String;
5    i : Integer;
6    Begin Writeln ('Donner un mot');
```

```
 7    Readln (ch);
 8     inv := '' ;
 9     FOR i := Length (ch) Downto 1 Do
10     inv_ch := inv_ch + ch[i];
11     IF ch = ch_inv Then Writeln (ch, ' est palindrome')
12     Else Writeln (ch, ' n''est pas palindrome');
13     end.
```

Chapitre 6

Les sous programmes: procédures et fonctions

Sommaire

6.1 Introduction

Dans un programme, surtout s'il est long, il peut arriver qu'on utilise le même bloc d'instructions plusieurs fois à des endroits différents. Dans ce cas il serait nécessaire, pour ces blocs d'instructions qu'on doit répéter dans différents endroits du programme, de créer un sous-programme (ou module) dont le code sera défini une seule fois, et que l'on appellera dans le corps du programme aux différents endroits souhaités.
Dans le langage Pascal, un sous programme peut être sous forme d'une procédure ou d'une fonction.

Exemple: le calcul de

$$\begin{pmatrix} n \\ k \end{pmatrix} = \frac{n! * k!}{(n-k)!}$$

nécessite le calcul de 3 factorielles ; on pourra écrire un sous programme (fonction) factorielle, et ensuite l'appeler 3 fois.

L'utilisation des sous programmes permet:

- d'assurer la lisibilité du programme.
- d'optimiser le nombre d'instructions du programme.
- de faciliter la maintenance, et la correction d'erreurs.

La structure d'un sous programme est la même qu'un programme, elle se compose d'un en-tête, d'une partie déclaration et d'une partie instructions qui commence par begin et se termine par end.
Le end du sous-programme est suivi d'un (;) et non pas d'un(.).
Les sous-programmes sont déclarés dans la partie déclaration du programme principal, juste avant le Begin du programme principal.
On distingue deux classes de sous programmes: les sous-programmes standards, ceux sont les procédures et les fonctions prédéfinies, donc reconnues par le langage, c'est le cas, par exemple, des procédures read, readln, write et writeln et des fonctions ABS, ORD, PRED, SQR et SQRT; et les sous-programmes non standards, qui nécéssitent une définition avant leur utilisation. C'est cette dernière classe qui fait l'objectif principal de ce chapitre.

6.2 Les variables locales

- Tout comme un programme, le sous programme contient ses propres variables, elles sont dites variables locales, ce sont les variables qui ne sont utiles que dans le sous programme et elles sont déclarées dans la partie des déclarations du sous programme.
- Une variable locale n'existe que pendant l'exécution de la procédure (fonction), et ne sert qu'à cette procédure (fonction).

6.3 Portée des variables

La portée d'une variable est l'ensemble de sous programmes qui peuvent utiliser cette variable.

- Les variables déclarées dans le VAR du programme principal sont appelées variables globales. Elles existent pendant toute la durée du programme et peuvent être utilisées par tous les sous programmes du programme principal.
- La portée d'une variable locale est uniquement le sous programme qui la déclare; le programme principal n'a jamais accès à une variable locale d'une procédure (fonction).
- Une procédure (fonction) n'a jamais accès à une variable locale d'une autre procédure (fonction).
- Lorsque le nom d'une variable locale est le même q'une variable globale, la variable globale est localement masquée pendant l'exécution de la procédure (fonction)(c.-à-d. dans la procédure, la variable globale devient inaccessible).

Exemple

```
1   PROGRAM local_global;
2   Uses wincrt;
3   VAR
4   ch: string;
5
6   PROCEDURE exemple_local;
7   VAR ch: string;
8   BEGIN
9   ch:= 'ch_local';
10  writeln ('var= ',ch);
11  END;
12
13  BEGIN
14  ch:= 'ch_global';
15  writeln ('var= ',ch);
16  exemple_local;
17  writeln ('var= ',ch);
18  END.
```

Après exécution, ce programme affiche :

var= ch_global
var= ch_local
var= ch_global

6.4 Les procédures

6.4.1 Définition

Une procédure est une séquence d'instructions de programme à laquelles on donne un nom qui est représenté par un identificateur.
En langage Pascal, on distingue les procédures sans paramètres et les procédures avec paramètres

6.4.2 Syntaxe de déclaration

1. Procédures sans paramètres

```
1  PROCEDURE  Nom_procedure;
2  VAR
3  {Déclaration des variables LOCALES (ou constantes locales) ;}
4  BEGIN
5  Liste des instructions;
6  END;
```

2. Procédures avec paramètres

```
1  PROCEDURE  Nom_procedure ( Liste_paramètres : Type_paramètres);
2  VAR
3  {Déclaration des variables LOCALES (ou constantes locales) ;}
4  BEGIN
5  Liste des instructions;
6  END;
```

La première ligne représente l'en-tête de la procédure. Liste_paramètres sert à échanger les données et les résultats avec le programme (sous programme) appelant.

Exemples:

1. Si on veut écrire un programme qui permet de sauter une ligne, il suffit d'appeler la procédure standard Writeln;
2. Si on veut écrire un programme qui permet de sauter 3 lignes consécutives, on appelle la procédure sans paramètres suivante:

```
1  Procedure sauter_3_lignes;
2  var
3   i: integer;
4  Begin
5  for i:=1 to 3 do
6  writeln;
7  end;
```

3. Si on veut écrire un programme qui permet de sauter n lignes consécutives, n est un entier donné par l'utilisateur, on appelle la procédure avec paramètres suivante:

```
1  Procedure sauter_nb_lignes(nb_lignes: integer);
2  var
3   i: integer;
4  Begin
5  for i:=1 to nb_lignes do
6  writeln;
7  end;
```

6.4.3 Appel d'une procédure

L'appel d'une procédure représente une instruction en elle même, comme c'est le cas des procédures read et write.

Syntaxe d'appel

1. Procédures sans paramètres:

```
1   Nom_procedure;
```

2. Procédures avec paramètres:

```
1   Nom_procedure ( Liste_paramètres );
```

Remarque:
Une procédure P1 peut être appelée depuis une procédure P2, mais il faut que la procédure P1 doit être déclarée avant la procédure P2. Aussi une procédure peut être appelée depuis elle même: c'est la récursivité.

6.5 Paramètres

- Les paramètres sont les variables par lesquelles un sous programme appelé peut communiquer avec le programme appelant.
- Les paramètres définis lors de la déclaration du sous programme sont appelés paramètres formels.
- Les paramètres utilisés lors de l'appel à un sous programme sont appelés paramètres effectifs.
- Pour appeler un sous-programme, il suffit donc de l'appeler par son nom, suivi de la liste des paramètres effectifs à lui transmettre.
- Lorsqu'il y a plusieurs paramètres dans un sous-programme, ils seront séparés par un " ; " dans la déclaration et par une " , " , sans préciser le type, lors de l'appel du sous programme dans le corps du programme.

6.6 Passage de paramètres

C'est le mécanisme de remplacement des paramètre formels d'une procédure par les paramètres effectifs, au moment de l'appel de la procédure.
Les paramètres effectifs doivent correspondre en ordre, en nombre et en type aux paramètres formels.
En Pascal, on distingue deux types de passage de paramètres : le passage par valeur et le passage par adresse.

6.6.1 Passage de paramètres par valeur

Ce type de passage de paramètres permet au programme appelant de transmettre une valeur à la procédure appelée. A l'appel, le paramètre effectif est une variable, une constante ou une expression. Le transfert de l'information est effectué du programme principal vers la procédure (transfert dans un seul sens). Ce type de passage ne permet pas de transmettre des résultats par l'intermédiaire d'un paramètre effectif, car toute modification est sans conséquent sur le paramètre effectif.
La déclaration d'un paramètre passé par valeur est la suivante:

```
1   PROCEDURE  Nom_procedure (Nom_paramètre : Type_paramètre);
```

Exemple

```
1   PROGRAM Passage_par_valeur;
2   Uses wincrt;
3   VAR
4   x:integer;
5
6   PROCEDURE Calcul(a:integer);
7   BEGIN
8   a:= 2*a+1;
9   writeln ('paramètre passé par valeur a = ',a);
10  END;
11
12  BEGIN
13  x:= 6;
14  writeln ('Valeur de x avant l'appel de la procédure Calcul, x= ',x);
15  Calcul(x);
16  writeln ('Valeur de x après l'appel de la procédure Calcul, x= ',x);
17  END.
```

Après exécution, ce programme affiche :

Valeur de x avant l'appel de la procédure Calcul, x= 6
paramètre passé par valeur a = 13
Valeur de x après l'appel de la procédure Calcul, x= 6

D'après le résultat, on remarque que l'appel de la procédure Calcul n'a aucun effet sur la variable x.

6.6.2 Passage de paramètres par adresse

Ce type de passage de paramètres permet au programme appelant de transmettre une valeur à la procédure appelée et vis-versa. A l'appel, le paramètre effectif est une variable uniquement (jamais une expression ou une constante). C'est l'adresse mémoire de la variable qui est transmise, non sa valeur. Ce type de passage est utilisé pour transmettre des résultats au programme principal ou à d'autres modules, car le transfert de l'information est effectué dans les deux sens, et toute modification du paramètre formel implique automatiquement la modification de la valeur du paramètre effectif.
La déclaration d'un paramètre passé par adresse est la suivante:

```
1   PROCEDURE  Nom_procedure (VAR Nom_paramètre : Type_paramètre);
```

C'est le mot-clé var qui dit si le passage se fait par valeur (pas de var) ou par adresse (présence du var).

Exemple

```
1   PROGRAM Passage_par_adresse;
2   Uses wincrt;
3   VAR
4   x:integer;
5
6   PROCEDURE Calcul(var a:integer);
7   BEGIN
8   a:= 2*a+1;
9   writeln ('paramètre passé par adresse a = ',a);
10  END;
11
12  BEGIN
13  x:= 6;
14  writeln ('Valeur de x avant l'appel de la procédure Calcul, x= ',x);
15  Calcul(x);
16  writeln ('Valeur de x après l'appel de la procédure Calcul, x= ',x);
17  END.
```

Après exécution, ce programme affiche :

Valeur de x avant l'appel de la procédure Calcul, x= 6
paramètre passé par adresse a = 13
Valeur de x après l'appel de la procédure Calcul, x= 13

D'après le résultat, on remarque que l'ajout du mot clé var dans l'entête de la procédure Calcul a entraîné le changement de la valeur de la variable x.

Le passage de paramètres peut être résumé dans ce petit tableau:

	Passage par valeur (Absence du mot clé **var**)	Passage par adresse (Présence du mot clé **var**)
paramètres données	oui	oui
paramètres résultats	non	oui

Quelques erreurs à éviter:

- Mettre un var quand il n'en faut pas → erreur à la compilation.

- Oublier le var quand il en faut → erreur à l'exécution.

Exemple
Appel d'une procédure Produit, qui permet de calculer le produit de deux nombres et de renvoyer le résultat au programme principal.

La ligne d'appel est la suivante: Produit (a-1, b+1, d)

1. Si la procédure est définit comme suit:

```
1   PROCEDURE Produit (var x : real; y : real; var z : real);
```

   Erreur à la compilation à cause du paramètre x, où une variable est attendue et c'est une expression qui est passée.

2. Si la procédure est définit comme suit:

```
1   PROCEDURE Produit (x, y, z : real);
```

   Erreur à l'exécution à cause du var oublié devant le paramètre z: d ne reçoit pas de résultat.

- Redéclarer un paramètre comme variable locale → erreur à la compilation.

Exemple

```
1   PROCEDURE Produit (x, y : real; var z : real);
2   VAR
3   x : real; {redéclaration du paramètre x : erreur}
4   BEGIN
5   z := x * y;
6   END;
```

6.7 Les fonctions

6.7.1 Définition

Une fonction est une séquence d'instructions de programme à laquelle on associe un nom spécifique représenté par un identificateur, un type qui spécifie le résultat fourni par cette fonction et des paramètres. Ainsi une fonction peut être vue comme une procédure qui ne retourne qu'un seul résultat au programme appelant. La notion de fonction en Pascal est assez semblable à celle de fonction en mathématiques.

6.7.2 Syntaxe de déclaration

```
1   FUNCTION  Nom_fonction (Noms_paramètre : Type_paramètre) : Type_resultat ;
2   VAR
3   {Déclaration des variables LOCALES (ou constantes via CONST) ;}
4   BEGIN
5   Liste des instructions;
6   Nom_fonction := Valeur_resultat ;
7   END;
```

6.7.3 Appel d'une fonction

L'appel d'une fonction se fait dans une expression, exactement comme une variable, mais suivie par les paramètres entre parenthèses, séparés par des virgules.

Remarques

- On doit préciser dans la définition de la fonction le type du résultat à retourner par la fonction.
- On doit affecter le résultat de la fonction au nom de la fonction avant la fin de la partie instructions de la fonction.
- Dans une fonction, le passage de paramètres se fait par valeur.
- Une fonction sans paramètres n'a aucun intérêt.

Exemple:

```
1   PROGRAM fonction;
2   Var
3    X: real;
4
5   FUNCTION affich(a : REAL):REAL ;
6   BEGIN
7   affich := 5*sqrt(a)+2;
8   END;
9
10  BEGIN
11  X := affich(3) ;
12  write(x);
13  END.
```

Le même exemple peut être écrit sans utiliser la variable globale x, en appelant la fonction affich() directement dans le write.

```
1   PROGRAM fonction;
2
3   FUNCTION affich(a : REAL):REAL ;
4   BEGIN
5   affich := 5*sqrt(a)+2;
6   END;
7
8   BEGIN
9   write(affich(3));
10  END.
```

6.8 Différences entre fonctions et procédures

Fonctions	Procédures
• Les fonctions ne peuvent avoir que des paramètres données. • Les fonctions ne peuvent communiquer qu'un seul résultat au programme appelant à travers une valeur de retour (et non à travers un paramètre) • Une fonction s'appelle à l'intérieur d'une instruction. L'instruction utilise la valeur retournée par la fonction.	• Les procédures peuvent avoir des paramètres résultats, données ou données/résultats. • Les procédures peuvent communiquer de 0 à plusieurs résultats au programme appelant à travers des paramètres résultats ou données/résultats. • L'appel d'une procédure représente une instruction en elle-même. On ne peut pas appeler une procédure au milieu d'une instruction

6.9 Exercices d'application

Exercice 1

Ecrire un programme pascal qui permet de calculer

$$\binom{n}{k} = \frac{n! * k!}{(n-k)!}$$

Exercice 2

Ecrire un programme Pascal qui contient une fonction qui teste si un nombre est positif ou négatif

Exercice 3

Ecrire un programme Pascal qui appelle une procédure qui fait l'échange du contenu de deux variables entières.

Solutions

Solution 1

```
1  PROGRAM factoriel;
2  Uses wincrt;
3  VAR
4  k,n:INTEGER;
5  fact: real;
6
```

```
7   FUNCTION factorielle (N : INTEGER) : INTEGER ;
8   VAR i,F : INTEGER ;
9   BEGIN
10  F:=1;
11  FOR i:=1 TO N DO
12  F:=F*i;
13  factorielle :=F;
14  END;
15
16  BEGIN
17  write('Entrer deux entiers n et k');
18  readln(n,k);
19  fact:= factorielle(n)/(factorielle(k)*factorielle(n-k)) ;
20  write(fact);
21  END.
```

Solution 2

```
1   Program pair_positif;
2   Uses Wincrt;
3   Var
4    a : Integer;
5    etat : boolean ;
6    Function positif ( x: Integer) : boolean ;
7    Begin if x < 0 then positif:= false
8              else positif:= true ;
9    End;
10  Begin
11  Writeln('Donner une valeur entière') ;
12  read(a) ;
13  etat:= positif(a) ;
14  if etat = true then writeln('Le nombre ', a , 'est positif')
15       else writeln('Le nombre ', a , 'est négatif') ;
16  end.
```

Solution 3

solution 1

```
1   PROGRAM echange_version1;
2   Uses Wincrt;
3   VAR
4    x1, x2, x3: integer;
5   PROCEDURE echange;
6   BEGIN
7   x3:= x1;
8   x1:=x2;
9   x2:=x3;
```

```
10  End;
11
12  BEGIN   { programme principal}
13  x1:= 10;
14  x2:= 4;
15  x3:= 15;
16  Writeln('valeurs de x1 et x2 avant appel de la procédure echange:');
17  Writeln('x1= ', x1,' x2= ',x2,' x3= ', x3);
18  echange;
19  Writeln('valeur après appel de la procédure echange:');
20  Writeln('x1= ', x1,' x2= ',x2,' x3= ', x3);
21  END.
```

solution 2

```
1   PROGRAM echange_version2;
2   Uses Wincrt;
3   VAR
4    x1, x2, x3: integer;
5   PROCEDURE echange;
6   var
7    aux: integer;
8   BEGIN
9   aux:= x1;
10  x1:=x2;
11  x2:=aux;
12  End;
13
14  BEGIN   { programme principal}
15  x1:= 10;
16  x2:= 4;
17  x3:= 15;
18  Writeln('valeurs de x1 et x2 avant appel de la procédure echange:');
19  Writeln('x1= ', x1,' x2= ',x2,' x3= ', x3);
20  echange;
21  Writeln('valeur après appel de la procédure echange:');
22  Writeln('x1= ', x1,' x2= ',x2,' x3= ', x3);
23  END.
```

solution 3

```
1   PROGRAM echange_version3;
2   Uses Wincrt;
3   VAR
4    x1, x2, x3: integer;
5   PROCEDURE echange(a,b: integer);
6   var
7    aux: integer;
8   BEGIN
```

```
 9   aux:= a;
10   a:=b;
11   b:=aux;
12   End;
13
14   BEGIN   { programme principal}
15   x1:= 10;
16   x2:= 4;
17   x3:= 15;
18   Writeln('valeurs de x1 et x2 avant appel de la procédure echange:');
19   Writeln('x1= ', x1,' x2= ',x2,' x3= ', x3);
20   echange(x1,x2);
21   Writeln('valeur après appel de la procédure echange:');
22   Writeln('x1= ', x1,' x2= ',x2,' x3= ', x3);
23   END.
```

solution 4

```
 1   PROGRAM echange_version4;
 2   Uses Wincrt;
 3   VAR
 4    x1, x2, x3: integer;
 5   PROCEDURE echange(VAR a,b: integer);
 6   var
 7    aux: integer;
 8   BEGIN
 9   aux:= a;
10   a:=b;
11   b:=aux;
12   End;
13
14   BEGIN   { programme principal}
15   x1:= 10;
16   x2:= 4;
17   x3:= 15;
18   Writeln('valeurs de x1 et x2 avant appel de la procédure echange:');
19   Writeln('x1= ', x1,' x2= ',x2,' x3= ', x3);
20   echange(x1,x2);
21   Writeln('valeur après appel de la procédure echange:');
22   Writeln('x1= ', x1,' x2= ',x2,' x3= ', x3);
23   END.
```

Question: quelle est la meilleure solution parmi ces différentes solutions proposées?

- Solution 1: il n'y a pas de paramètres, ni de variables locales, les variables globales x1 et x2 sont bien échangées
  Inconvénients: la valeur de la variable x3 est modifiée
  La procédure n'est pas générale, car elle ne permet que l'échange des deux variables x1 et x2, pour échanger d'autres variables, il faudrait une autre procédure.

- Solution 2: il n'y a pas de paramètres, mais une variable locale, les variables globales x1 et x2 sont bien échangées.
  La variable locale a permi de ne pas modifié la valeur de x3, mais la procédure reste toujours limitée à l'échange de X1 et x2.

- Solution 3: il y a deux paramètres par valeur et une variable locale.
  Les variables associées aux paramètres par valeur ne sont pas modifiées, donc les variables x1 et x2 ne sont pas échangées.

- Solution 4: il y a deux paramètres par adresse et une variable locale.
  Les variables x1 et x2 sont bien changées.
  La procédure peut être utilisée pour échanger les valeurs d'autres variables.

Donc la dernière solution et la meilleure solution.

Série d'exercices (avec solutions proposées)

Exercice 1

Ecrire les expressions suivantes en langage Pascal:

(1) $(2x^2 - 3)\frac{z^{2+14}}{\sqrt{7y+15}}$

(2) $\frac{xy^3}{y-7z}$

(3) $\frac{(z^2-5)\sqrt{x}}{5y-3}$

Utiliser les fonctions du langage Pascal qui calculent le carré et la racine carrée.

Exercice 2

1- Quelles erreurs ont été commises dans chacune des instructions suivantes:

1. if a<b then x := x+1 ; else x := x-l
2. if a< b then x := x+1 ; y := b end else x := x-1 ; y := a end
3. if n := 0 then p := 1

2- Exécuter les programmes suivants. Lesquels fonctionnent ? Proposer des modifications pour ceux qui ne fonctionnent pas.

(1)

```
1   PROGRAM P1 ;
2   Uses wincrt;
3   VAR x,z :integer ;
4   y :real;
5   BEGIN
6   x :=7 ;
7   y :=4 ;
8   z :=2*x+y ;
9   writeln(z) ;
10  END.
```

(2)

```
1   PROGRAM P2 ;
2   Uses wincrt;
3   VAR x,z :real ;
4   y :integer ;
5   BEGIN
6   x :=5 ;
7   y :=2 ;
8   z :=x*y ;
9   writeln('z') ;
10  END.
```

(3)

```
1   PROGRAM P3 ;
2   Uses wincrt;
3   VAR x,y,z :integer ;
4   BEGIN
5   x :=12 ;
6   y :=4 ;
7   z :=x/y ;
8   writeln(z) ;
9   END.
```

(4)

```
1   PROGRAM P4 ;
2   Uses wincrt;
3   VAR x,y :real ;
4   BEGIN
5   x :=1 ;
6   y :=2*x ;
7   z :=y-3 ;
8   writeln(z) ;
9   END.
```

Exercice 3

1. Ecrire un programme, qui demande à l'utilisateur d'entrer un nombre x, et qui affiche son carré, son cube,sa racine carrée, et sa valeur absolue (sans utiliser la fonction ABS du langage PASCAL).

2. Ecrire un programme qui demande à l'utilisateur son année de naissance, et qui affiche son âge.

Exercice 4

Ecrire un programme qui demande deux nombres à l'utilisateur et l'informe ensuite si le produit est négatif, positif ou nul.

Exercice 5

Ecrire un programme qui demande le temps, exprimé en secondes, et qui le transforme en h : m : s. On utilisera les fonctions prédéfinies div (division entière) et mod (reste entier) du langage pascal.

Exercice 6

Écrire un programme qui lit deux notes d'examen et leur coefficient, puis affiche la moyenne en précisant "ajourné" si la moyenne est inférieure à 10, "admis" dans le cas contraire.

Exercice 7

Ecrire un programme, qui demande 3 réels a; b et c à l'utilisateur, puis qui calcule les solutions de l'équation $ax^2 + bx + c = 0$.

Exercice 8

Ecrire un programme qui affiche les nombres entiers de 1 à 50, en utilisant les boucles : for , while, repeat.

Exercice 9

Ecrire un programme qui calcule la somme des nombre impairs et la somme des nombres pairs compris entre 1 et 20.

Exercice 10

Ecrire un programme qui affiche les diviseurs d'un nombre entier N.

Exercice 11

Ecrire un programme qui permet de déterminer et d'afficher le successeur et le prédécesseur d'un caractère c donné par l'utilisateur.

Exercice 12

Ecrire un programme qui permet d'afficher la première et la deuxième position d'un caractère c dans une chaîne de caractères ch.
Exemple :
Pour ch= 'programmation':

- Si c = 'm', le programme affichera :
  1ière position =7, 2ième position = 8.
- Si c = 'a', le programme affichera :
  1ière position =6, 2ième position = 9.
- Si c = 'g', le programme affichera :
  1ière position =4, 2ième position = 0.
- Si c = 'k', le programme affichera :
  1ière position =0, 2ième position = 0.

Exercice 13

Ecrire un programme qui permet de chercher la plus grande valeur dans un tableau T de 40 entiers, ainsi que le nombre d'occurrence de cette valeur et l'indice de la première occurrence.

Exercice 14

Ecrire un programme qui permet de remplacer toute occurrence d'une sous-chaîne mot1 par la sous-chaîne mot2 dans une chaîne de caractères nommée texte.

Exercice 15

Ecrire un programme Pascal qui permet de lire une phrase, de supprimer les espaces superflus (laisser un seul espace entre deux mots)et de l'afficher renversée. La phrase commence, obligatoirement, par une lettre ne se termine pas par un espace.

Exemple :
La phrase initiale:
'Cours de programmation'
Résultat :
'programmation de cours'

Exercice 16

Ecrire un programme qui permet de lire un tableu de 15 entiers, puis pour chaque élement du tableau ne garde que sa première occurrence, en remplaçant les autres par zéro, et regrouper les éléments non nuls au début du tableau.

Exercice 17

Ecrire un programme qui permet de lire une chaîne de caractères de longueur minimale 3 et l'affiche sous la forme d'un triangle comme indiqué ci-dessous.
Exemple : Si la chaîne saisie est "INTERNET", on aura : I
IN
INT
INTE
INTER
INTERN
INTERNE
INTERNET

Exercice 18

Ecrire un programme qui permet de calculer le produit de deux matrices A (2,3) et B(3.4).

Solutions

Solution 1

(1) (2*sqr(x)-3)*(sqr(z)+1)/sqrt(7*y=15)

(2) x*y*sqr(y)/(y-7*z)

(3) (sqr(z)-5)*sqrt(x)/(5*y-3)

Solution 2

1-

1. if $a < b$ then x := x+1 ; else x := x-l [Ne mettez jamais le point virgule (;) avant ELSE]
2. if $a < b$ then x := x+1 ; y := b end else x := x-1 ; y := a end [Chaque END doit avoir un BEGIN et après le dernier end, on ajoute un point virgule], l'écriture correcte est:

```
1   if a<b then
2    begin
3     x := x+1 ;
4     y := b
5    end
6    else
7    begin
8     x := x-1 ;
9     y := a
10   end;
```

3. if n := 0 then p := 1 [Ne pas confondre := avec le = réservé aux données constantes et aux symboles de comparaison]. L'écriture juste est : if n = 0 then p := 1;

2-

(1) Le programme ne fonctionne pas, le problème se trouve dans l'instruction z :=2*x+y , le z est un entier par contre le y est un réel
Le programme correct est :

```
1   PROGRAM test1 ;
2   Uses wincrt;
3   VAR x :integer ;
4   y ,z:real;
5   BEGIN
6   x :=7 ;
7   y :=4 ;
8   z :=2*x+y ;
9   writeln(z) ;
10  END.
```

(2) Ici le programme n'affiche pas la valeur de z mais tout simplement il affiche le caractère z . au lieu d'écrire writeln('z') ;nous écrivons writeln(z) ;

(3) L'opérateur / est utilisé dans le cas d'une division réelle mais dans notre cas x et y sont des entier, l'écriture juste est : z :=x div y

(4) Dans la partie instructions , nous trouvons l'instruction : z := y-3 ; mais la variable z n'est plus déclarée. Le programme correct est :

```
1  PROGRAM test4 ;
2  Uses wincrt;
3  VAR x,y,z :real ;
4  BEGIN
5  x :=1 ;
6  y :=2*x ;
7  z :=y-3 ;
8  writeln(z) ;
9  END.
```

Solution 3

1-

```
1  PROGRAM calcul;
2  Uses wincrt;
3  VAR x,cube,carre val_abs : integer;
4      r_carre:real;
5  BEGIN
6  writeln('Entrer un nombre :');
7  readln(x);
8  carre:=sqr(x);
9  r_carre:=sqrt(x);
10 cube:=x*carre;
11 If x > 0 then val_abs := x
12            else val_abs := -x;
13 writeln('le carré de ',x,' est ', carre);
14 writeln('la racine carrée de ',x,' est ', r_carre);
15 writeln('le cube de ',x,' est ', cube);
16 write('La valeur absolue de ', x, ' est ', val_abs);
17 end.
```

2-

```
1  PROGRAM age;
2  Uses wincrt;
3  VAR an_nais,age : integer;
4  BEGIN
5  writeln('Entrer votre année de naissance(exemple:1980) :');
6  readln(an_nais);
7  age:=2018-an_nais;
8  writeln('Vote âge est: ', age);
9  end.
```

Solution 4

```
1   PROGRAM produit;
2   Uses wincrt;
3   VAR x,y,p : integer;
4   BEGIN
5   writeln('Entrer deux nombres :');
6   readln(x,y);
7   p:=x*y;
8   if p=0 then write('produit nul')
9    else
10      if p>0 then write('produit positif')
11             else write('produit négatif');
12  END.
```

Autre solution:

```
1   PROGRAM produit;
2   Uses wincrt;
3   VAR x,y : integer;
4   BEGIN
5   writeln('Entrer deux nombres :');
6   readln(x,y);
7   if (x=0)or(y=0) then   writeln('produit nul')
8   else if ((x>0)and(y>0))or((x<0)and(y<0)) then writeln('produit positif')
9                                              else writeln('produit négatif');
10  END.
```

Solution 5

```
1   PROGRAM temps;
2   VAR T,s,m,m1,h : integer;
3   BEGIN
4   write('Entrer le temps exprimé en secondes :');
5   readln(T);
6   s:=T mod 60;
7   m1 := T div 60;
8   (* On a transformé notre temps T en minutes et secondes ,
9    c'est donc ce nombre m1 de minutes qu'il faut maintenant transformer
10                    en heures et minutes *)
11  m:= m1 mod 60 ;
12  h:= m1 div 60 ;
13  writeln('Le temps est de ',h, ' heures ', m, ' minutes et ', s, ' secondes.');
14  END.
```

Autre solution: T=h*3600+m*60+s

```
1   PROGRAM temps;
2   Uses wincrt;
```

```
3   VAR T,s,m,h : integer;
4   BEGIN
5   write('Entrer le temps exprimé en secondes :');
6   readln(T);
7   h:=T div 3600;
8   m:=(T mod 3600 ) div 60 ;
9   s:=(T mod 3600) mod 60;
10  writeln('Le temps est de ',h, ' heures ', m, ' minutes et ', s, ' secondes.');
11  END.
```

Solution 6

```
1   PROGRAM moyenne_etud;
2   Uses wincrt;
3   VAR n1,n2,moy : real;
4       c1,c2: integer;
5   BEGIN
6   writeln('Entrer la première note: /20 ');
7   readln(n1);
8   writeln('Entrer le coefficient du premier module:  ');
9   readln(c1);
10  writeln('Entrer la deuxième note: /20 ');
11  readln(n2);
12  writeln('Entrer le coefficient du deuxième module ');
13  readln(c2);
14  moy:= (n1*c1+n2*c2)/(c1+c2);
15  if moy>=10 then writeln('moyenne= ',moy,' => étudiant admis')
16             else writeln('moyenne= ',moy,' =>étudiant ajourné');
17  end.
```

Solution 7

```
1   PROGRAM eq_2_deg;
2   Uses wincrt;
3   VAR
4           a,b,c,x1,x2, delta: real;
5   Begin
6    Writeln('************************************************************');
7    writeln('***      Résolution d'une équation du 2ième degré        ***');
8    Writeln('************************************************************');
9    Writeln('entrer les valeurs de a, b et c:');
10   Read(a,b,c);
11   if (a=0) then if (b=0) then if (c=0)then write('tout nombre est une solution')
12                          else write('solution impossible')
13            else ('La solution est: ', -c/b)
14            else Begin
15            delta:= sqr(b)-4*a*c;
16           if delta< 0 then write('solution impossible')
```

```
17                              else if delta=0 then
18                              begin
19                               writeln('solution double');
20                               write ('x=', -b/2*a);
21                               end
22                               else
23                               begin
24                                     writeln('2 solutions :');
25                                     x1= (-b-sqrt(delta))/2*a;
26                                     x2= (-b+sqrt(delta))/2*a;
27                                     write ('x1= ',x1,' ,x2= ',x2);
28                                     end;
29   end.
```

Solution 8

1. For-do:

```
1   PROGRAM affich_entier;
2   Uses wincrt;
3   VAR
4     i: integer;
5   Begin
6    for i:=1 to 50 do
7    writeln('i= ', i);
8   end.
```

2. while-do:

```
1   PROGRAM affich_entier;
2   Uses wincrt;
3   VAR
4     i: integer;
5   Begin
6   i:=1;
7    while (i<=50) do
8   begin
9    writeln('i= ', i);
10  i:=i+1;
11  end;
12  end.
```

3. repeat-until:

```
1   PROGRAM affich_entier;
2   Uses wincrt;
3   VAR
4     i: integer;
```

```
5   Begin
6   i:=1;
7    repeat
8    writeln('i= ', i);
9    i:=i+1;
10  until (i>50);
11  end.
```

Solution 9

```
1   PROGRAM somme_pair_impair;
2   Uses wincrt;
3   VAR
4           i, s_i,s_p: integer;
5   Begin
6   for i:=1 to 20 do
7   if (i mod 2)=0 then s_p:=s_p + i
8                           else s_i:=s_i + i;
9   writeln('la somme des nombres pairs compris entre 1 et 20 est:',s_p);
10  write('la somme des nombres impairs compris entre 1 et 20 est:',s_i);
11  end.
```

Solution 10

```
1   PROGRAM diviseurs;
2   Uses wincrt;
3   VAR
4     n,i: integer;
5   begin
6           read(n);
7   for i:=1 to (n div 2) do
8   if (n mod i)=0 then writeln(i);
9   end.
```

Solution 11

```
1   Program SUCC_PRED;
2   Uses WinCrt;
3   Var
4   c,s,p:char;
5   Begin
6   Writeln('Tapez un caractère');
7   Readln(c);
8   s:= succ(c); p:= pred(c);
9   Writeln('Le successeur de "',c,'" est "',s,'"');
10  Writeln('Le prédécesseur de "',c,'" est "',p,'"');
11  End.
```

Solution 12

```
1   Program POSITION_1_2;
2   Uses WinCrt;
3   Var
4   c:char; ch:string; p1, p2:integer;
5   Begin
6   Writeln('Tapez un caractère'); Readln(c);
7   Writeln('Tapez une chaîne de caractère'); Readln(ch);
8   p1:= pos(c,ch);
9   p2:= p1 + pos(c, copy(ch,p1 + 1, length(ch) - p1));
10  Writeln('La première position de "',c,'" dans "',ch,'" = ',p1);
11  Writeln('La deuxième position de "',c,'" dans "',ch,'" = ',p2);
12  End.
```

Solution 13

```
1   Program EXISTANCE;
2   Uses WinCrt;
3   Type
4   TAB = Array[1..40] of integer;
5   Var
6   T:TAB;
7   max,i,occ, ind :integer;
8   Begin
9   For i := 1 to 40 Do
10  Readln(T[i]);
11  max:=T[1];
12  For i := 2 to 40 Do
13  if T[i]> max then max:=T[i];
14  i:=40;
15  occ:=0;
16  For i:=40 downto 1 do
17  if T[i]=max then begin
18                   occ:=occ+1;
19                   ind:=i;
20                   end;
21  Writeln('La valeur maximale =', max);
22  Writeln('Elle existe ',occ,'  fois dans T');
23  Writeln('Elle se trouve à l'indice: ', i);
24  End.
```

Solution 14

```
1   Program rechercher_remplacer;
2   Uses WinCrt;
3   Var
4   chaine,mot1,mot2:string;
```

```
p,l:integer;
Begin
Writeln('Tapez votre chaine');
Readln(chaine);
Writeln('Tapez le mot rechercher');
Readln(mot1);
Writeln('Tapez le mot à remplacer');
Readln(mot2);
p:=pos(mot1,texte);
l:=length(mot1);
While p <> 0 Do
Begin
Delete(chaine,p,l);
Insert(mot2,chaine,p);
p:=pos(mot1,chaine);
End;
Writeln('Chaine après modIfication: ',chaine);
End.
```

Solution 15

```
Program RENVERSEE;
Uses WinCrt;
Var
phrase, ph:string;
Function supprimer(ch:string):string;
Var
pt, l,i :integer;
Begin
l:=length(ch); i:=1;
while (i<= l-1) do
begin
if (ch[i]=' ' ) and (ch[i+1]=' ' )then
begin
delete(ch,i+1,1);
l:=length(ch);
end
else
i:=i+1;
end;
supprimer:=ch;
end;

Function renverser (ch:string):string;
Var
pt:integer;
ch1:string;
Begin
```

```
28          ch1:='';
29   While (pos(' ',ch) <> 0) Do
30   Begin
31   pt:= pos(' ',ch);
32   ch1:= copy(ch,1,pt-1)+ ' ' + ch1;
33   delete(ch,1,pt);
34   End;
35   ch1:= ch + ' ' + ch1;
36   renverser:=ch1;
37   End;
38   Begin
39   Writeln('Tapez votre phrase');
40   Readln(phrase);
41   ph:=renverser(supprimer(phrase));
42   Writeln(ph);
43   End.
```

Solution 16

```
1    Program ELEMENTS_DIFFERENTS;
2    Uses WinCrt;
3    Type
4    TAB = Array[1..15] of integer;
5    VAR
6    T:TAB;
7    Procedure lire_tableau (Var vec:TAB);
8    Var
9     i:integer;
10   Begin
11   For i := 1 To 15 Do
12   Readln(vec[i]);
13   End;
14   Procedure garder (Var vec:TAB);
15   Var
16    i,j:integer;
17   Begin
18   For i:=1 To 14 Do
19   For j:= i+1 To 15 Do
20   If vec[i] = vec[j] Then vec[j]:=0;
21   End;
22   Procedure regrouper (Var vec:TAB);
23   Var
24    vec_r:TAB;
25    ind,i:integer;
26   Begin
27   ind:=1;
28   For i:= 1 To 15 Do
29   If vec[i] <> 0 Then
```

```
30  Begin
31  vec_r[ind]:=vec[i];
32  ind:=ind+1;
33  End;
34  if ind<15 then
35  For i:=ind to 15 do
36  vec_r[i]:=0;
37  vec:=vec_r;
38  End;
39  Procedure afficher(vec:TAB);
40  Var
41   i:integer;
42  Begin
43  For i:= 1 To 15 Do
44  Write(vec[i], ' ');
45  End;
46  Begin
47  lire_tableau(T);
48  garder(T);
49  regrouper(T);
50  afficher(T);
51  End.
```

Solution 17

```
1   Program TRIANGLE;
2   Uses WinCrt;
3   Var
4   ch:string;
5   Procedure lire (Var ch:string);
6   Begin
7   Repeat
8   Writeln ('Tapez une chaîne'); Readln (ch);
9   Until (length (ch) >= 3);
10  End;
11  Procedure afficher (ch:string);
12  Var i,l:integer;
13  Begin
14  For i:= 1 To length (ch) Do Writeln (Copy (ch,1,i));
15  End;
16  Begin
17  lire (ch);
18  afficher (ch);
19  End.
```

Solution 18

```
1   Program matrice ;
```

```
2   uses wincrt;
3   var
4   m1 :array[1..2,1..3] of integer ;
5   m2 :array[1..3,1..4] of integer ;
6   mat :array[1..2,1..4] of integer ;
7           i, j, k : integer ;
8   begin
9   {Lecture de la première matrice:}
10  for i := 1 to 2 do
11  for j := 1 to 3 do
12  readln (m1[i, j]);
13
14  {Lecture de la deuxième matrice:}
15  for i := 1 to 3 do
16  for j := 1 to 4 do
17  readln (m2[i, j]);
18
19  {Calcul du produit:}
20  for i := 1 to 2 do
21  for j:=1 to 4 do
22  for k:=1 to 3 do
23  mat[i,j]:= mat[i,j]+ m1[i,k]*m2[k,j];
24  {Affichage du résultat:}
25  for i:=1 to 2 do
26  Begin
27  for j:=1 to 4 do
28  write(mat[i,j],'     ');
29  writeln;
30  end;
31  end.
```