Republique Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientique
Université de 8 Mai 1945 Guelma

Faculté des Mathématiques, Informatique et Sciences de la Matière

Département d'informatique

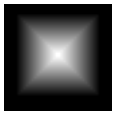
Polycopié

INFORMATIQUE 2

Travaux pratiques en langage Pascal

Niveau : 1ère ANNÉE ST

Présenté par : M. Derdar Salah



Année universitaire 2022-2023

Préface

Ce polycopié de travaux pratiques regroupe un rappel sur les notions de base de la programmation en langage Pascal. Deux objectifs à atteindre par ce polycopié: premièrement, il s'adresse aux étudiants et aux enseignants chargés du cours, travaux dirigés ou travaux pratiques et plus particulièrement aux étudiants de première année sciences et technologie (ST). Deuxièmement, il est rédigé dans un style simple, clair et riche d'exemples, que nous souhaitons compréhensible. Ce polycopié est enseigné pendant pratiquement une dizaine d'années par un groupe d'enseignants du département d'informatique de la faculté des Mathématiques, d'informatique et sciences de la matière (MISM). Le présent polycopié est scindé en deux parties. La première partie illustre l'essentiel sous forme de rappel sur l'algorithmique avec la programmation en Pascal qui comporte les tableaux 1- et 2-dimensions, les fonctions et les procédures et enfin les enregistrements et les fichiers. La deuxième partie est consacrée aux travaux pratiques concus pour aboutir à réaliser des solutions aux exercices en code Pascal. Ces derniers englobent l'ensemble des exercices traités, en séance de TP du module pendant le deuxième semestre, par les étudiants de première année ST pendant les années universitaires de 2018 à 2023. Ces séries d'exercices couvrent toutes les sections présentées en cours. Pour ce faire, nous allons proposer un ensemble de séries d'exercices traités et nous avons fait appel au langage Pascal qui est un langage facile à manipuler par un étudiant débutant et possède une structure très proche de celle du langage algorithmique.

L'objectif de ce polycopié est de construire un support qui fournit un rappel sur les notions de base enseignées pendant le deuxième semestre pour guider les étudiants dans l'apprentissage de la programmation en Pascal. Ce même support fournit aussi aux étudiants un manuel pratique et riche des exercices dont la solution se traduit par des codes en Pascal bien commentés.

A la fin, nous tenons à remercier Zemmouchi Fares Mounir, chargé du cours du module, avec lequel nous avons enseigné la matière en tant que chargé de TP au sein de la faculté de sciences et technologie (ST). Nos remerciements vont également à tous les enseignants de département d'informatique, ayant assuré le TP durant les dernières années universitaires pour bien munir cet effort académique et pour la confiance qu'ils nous ont accordé afin de gérer et enseigner le module d'algorithmique. Nos remerciements vont également aux responsables de la faculté des sciences et technologie (ST), ayant accepté d'enseigner ce module durant ces dernières années.

Les séries de TP englobent un ensemble d'exercices de base permettant à l'étu-

diant de se familiariser avec la programmation Pascal et sont réparties comme suit:

- série 1 traite les problèmes à propos des tableaux, à savoir les vecteurs et les matrices et aussi les chaînes de caractères.
- série 2 porte sur les procédures et les fonctions.
- série 3 porte sur les enregistfrements et les fichiers des deux types : texte et élément.

La bibliographie est rédigée soigneusement avec de sources bibliographiques très variées. Enfin, j'espère que ce polycopié pourra apporter un plus et une assistance pédagogique aux enseignants chargés de cours, TD ou TP, ainsi qu'aux étudiants dans leur formation touchant la programmation en langage informatique.

Dédicaces

Ce polycopié est dédié à la mémoire de mon père et tous les membres de ma famille.

Table des matières

Liste des tableaux

Table des figures

Chapitre 1

INTRODUCTION

Nous présentons dans ce chapitre les piliers liées à l'informatique. On commence par montrer et expliquer les origines de l'algorithmique et les notions de base qui conduisent à concevoir un futur programme informatique.

1.1 Généralités

L'informatique est la science du traitement automatique de l'information. Apparue au milieu du 20ème siècle, elle a connu une évolution extrêmement rapide. A sa motivation initiale qui était de faciliter et d'accélérer le calcul, se sont ajoutées de nombreuses fonctionnalités, comme l'automatisation, le contrôle et la commande de processus, la communication ou le partage de l'information. Le cours d'architecture des ordinateurs expose les principes de base du traitement et montre le codage binaire de l'information. La mise en oeuvre de représentation de l'information ert sin traitement s'appuient sur deux profils distincts, le matériel et le logiciel. Le matériel (hardware) correspond à l'aspect concret du système : unité centrale, mémoire, organes d'entrées-sorties, etc ... Le logiciel (software) correspond à un ensemble d'instructions, appelé programme, qui sont contenues dans les différentes mémoires du système et qui définissent les actions effectuées par le matériel.

Ci-dessous quelques évènements marquants l'évolution et le développement du domaine de technologie de l'information depuis l'apparition de l'ordinateur (Voici les dates des principaux événements qui ont marqué l'histoire du micro-ordinateur depuis 1980):

- 780-850: Abu Abdullah Muhammad bin Musa Al-Khawarizmi.
- 1854 : Boole publie "An Investigation Into the Laws of Thought", ouvrage fondateur de l'algèbre de Boole.
- 1946 : premier ordinateur l'ENIAC.
- 1947 : apparition du transistor (laboratoires Bell Téléphone).

- 1958 : Jack St. Clair Kilby invente le circuit intégré.

- 1964 : lancement de la série 360 d'ordinateurs d'IBM; ordinateurs compatibles entres eux.

- 1971 : commercialisation du Intel 4004; premier micro processeur Intel (4 Bits, 108 KHz, 60000 instructions par seconde, composé de 2300 transistors en technologie de 10 microns).

- 1973 :lancement du mini-ordinateur multitâches (temps-réel) et multi-utilisateur, le HP 3000, par Hewlett-Packard.

- 1981 à nos jours : c'est l'âge des micro-ordinateurs et des super calculateurs.
- Août 1981 : IBM lance le PC qui rend le Star obsolète.
- Janvier 1984 : Apple lance le Macintosh, incarnation réussie de l'ordinateur personnel conçu par le PARC.
- 1982 : Compaq commercialise le premier micro-ordinateur portable qui pese 15 kg.

- 1983 : Apple lance le Lisa, premier ordinateur possédant une interface graphique (menus déroulants, fenêtres, corbeille etc.).

- La norme IEEE 802.3 pour les réseaux locaux Ethernet est publiée. C'est le début de la généralisation des réseaux locaux dans les entreprises.

- Apple commercialise le Macintosh, qui apparaîtra comme le grand concurrent du PC.

- 1985 : IBM lance le PC AT qui a un grand succès.

- IBM annonce le réseau Token Ring, réseau local qui concurrencera Ethernet dans les entreprises industrielles.

- Intel lance le processeur 80386 à 16 MHz qui améliore de façon significative la puissance du PC.

  - Microsoft livre Windows 1.0 avec un interface graphique aux utilisateurs de PC.

- 1986 : les bases de données sur PC se développent avec dBASE dŠAshton et Tate.

- Compaq lance le premier PC 386.

- 1987 : les PC 386 détrônent les PC AT.

- IBM lance la série PS/2 et le système d'exploitation OS/2.
- Apple lance le Mac II.
- 1989 : Ethernet 10BaseT démarre. C'est l'année des réseaux locaux de PC.
- Intel annonce le processeur 486.
- Motorola lance le processeur 68040.
- Apple lance les Macs bas de gamme : Classic, LC et IIsi.
- 1991: Windows est en position de monopole, OS/2 disparaît de la scène.
- Naissance du World Wide Web : Tim Berners-Lee, à Genève, monte le premier serveur Web.
- 1993 : Début du déploiement du Pentium.
- Intel lance le processeur 60 MHz Pentium, Apple sort le Newton, Novell annonce NetWare 4.0, Lotus Notes 3.0 démarre, et Microsoft lance Windows NT.
- 1994 : L'architecture client/serveur prend pied sur le marché.
- En août, Microsoft livre Windows 95 et Intel lance le Pentium Pro à 150-200 MHz.

1.2 Informatique

1.2.1 L'histoire de la programmation informatique

C'est **Abu Abdullah Muhammad bin Musa Al-Khawarizmi** (780-850) le mathématicienn perse que son nom fait l'origine de l'appellation de l'algorithmique (ou Al-Khawarizmi). Aussi, deux evenements historiques qui ont caractérisé le début le domaine d'informatique :

- 1642 : l'invention de la Pascaline par Blaise Pascal.
- 1821 : invention de la machine à différences par Charles Babbage.

Le premier programme informatique de l'histoire revient à **Ada Lovelace** ou son èpoque représente le démarrage officiel de l'évolution de la programmation informatique:

Ada Lovelace, première programmeuse de l'histoire En 1840, Ada Lovelace (mathématicienne) nomme le processus logique d'exécution d'un programme : Algorithme, en l'honneur à Al-Khawarizmi. L'origine de la programmation vient en premier lieu de la programmeuse Ada Lovelace. Très connue dans le monde anglo-saxon. Pourtant, Ada Lovelace a inventé le premier langage de programmation en 1843.

La scientifique travaillait en ce temps sur une machine analytique de Charles Babbage qui n'est autre que l'ancêtre de nos ordinateurs. Elle développe alors le premier algorithme qu'elle écrit sur papier puisqu'elle ne pouvait pas l'écrire directement sur machine.

Fortran, Cobol, les prémices des langages de programmation Ci-dessous quelques évènements marquants l'évolution et le développement du domaine de technologie de l'information depuis l'apparition de l'ordinateur (Voici les dates des principaux événements qui ont marqué l'histoire du micro-ordinateur depuis 1980):

- Après Ada Lovelace, c'est notamment Plankalkül (ou plan de calcul), développé par Konrad Zuse entre 1944 et 1945, qui sera considéré comme le véritable premier code de programmation abouti. Ce langage servait alors à stocker des morceaux de code afin de réaliser certaines opérations sans avoir à recopier le code à chaque fois.

- Après quelques années, en 1949, le premier langage HLL (Langue de haut niveau) a été développé par un certain John McCauley et mis au point par William Schmitt.

- En 1950 : invention de l'assembleur à l'université de Cambridge.

- En 1957, John Backus crée le langage Fortran avec son équipe IBM. Fortran est concu pour réaliser les calculs scientifiques complexes. Fortran est notamment considéré comme le langage de programmation le plus vieux qui est encore utilisé jusqu'à nos jours.

- Une année plus tard, le langage algorithmique a été mis au point. ALGOL a notamment servi de base pour développer certains langages à savoir les langages de programmation C, C++ ou encore Java.

- En 1958, LISP rejoint la liste des langages de programmation. Tout comme Fortran, LISP est encore utilisé de nos jours par certaines entreprises. Il se rapproche notamment des langages plus connus Python et Ruby.

- En 1959 arrive COBOL (Common Business Oriented Language) inventé par Grace Murray Hopper. Le langage COBOL est encore utilisé de nos jours, notamment dans le domaine bancaire.

- En 1968 : apparition du Langage PASCAL, créé par Niklaus Wirth.

Les années 70 et 80 et les langages de programmation - Au début des années 1970, le programmeur Niklaus Wirth a développé le langage Pascal. La même personne a égalemet développé les langages Euler et Algol.

- En 1971„ Alan Kay et son équipe mettent au point la première version de Smalltalk, langage de programmation orienté objet qui influencera C++ et Java.

- En 1972, Dennis Ritchie a développé le langage C aux laboratores de Bell Telephone. Afin de développer son propre langage de programmation, ce dernier est basé sur son ancêtre le langage appelé B. Les langages de programmation modernes comme C#, Java, JavaScript, PHP, Python ou encore Perl utilisent le langage C comme base.

- En 1972, Raymond Boyce et Donald Chamberlain ont développé le langage SQL, un langage pour lire les bases de données.

- Au début des années 80, le langage Ada est alors mis au point par le département de la défense des Etats-Unis en hommage à la première programmeuse de l'histoire. Ce langage s'appuie au départ sur le langage de programmation Pascal.

- En 1983, le langage C++ a vu le jour par Bjarne Stroustrup. En modifiant le langage C, le langage C++ est devenu l'un des langages de programmation les plus utilisés dans le monde. Le langage C++ est notamment utilisé pour MS Office ou encore Adobe Photoshop.

- En 1987, Larry Wall a créé le langage Perl. Ce langage de programmation de haut niveau sert pour développer les éditeurs de traitement de texte, la mise au point des applications de base de données et la programmation réseau.

L'arrivée des langages modernes et d'internet en 90 - L'arrivée puis la démocratisation d'internet a bousculé les codes de la programmation. À partir des années 1990, plusieurs langages ont continué dŠapparaître à l'image de Python, né en 1991. Développé par Guido Van Rossum et nommé d'après les célèbres ńMonty Pythonż, le langage Python est aujourd'hui utilisé par des grosses têtes comme Google ou encore Spotify. Il fait partie des langages de programmation incontournables dans le milieu.

- La même année, Visual Basic est développé par Microsoft. Ce codage permet de visualiser le codage de manière graphique et ainsi de prendre une partie de code pour la déplacer ou la modifier, d'où le terme Visual Basic. Microsoft utilise ce codage notamment pour Word ou encore Excel. Il n'est cependant pas le plus répandu des codes informatiques.

- En 1993, Ruby est développé par Yukihiro Matsumoto. Ce langage de programmation de haut niveau s'inspire d'autres comme comme Perl ou Ada, et permet de développer des applications web comme Twitter mais aussi Groupon.

- Java fait également partie des langages de programmation apparus dans les années 90 et toujours très utilisé aujourdŠhui. Créé par James Gosling, célèbre programmeur, le langage Java est utilisé dans presque tous nos outils technologiques.

- En 1995, le langage PHP révolutionne internet. Développé par Rasmus Lerdorf, le langage PHP est notamment celui qui permet de créer des pages web dynamiques. Autrement dit, des pages avec de l'interactivité. Le PHP a changé notre façon de créer des sites web. Parmi les grands noms qui utilisent le langage PHP dans leur développement, on retrouve notamment Facebook, Wikipedia, WordPress ou encore le CMS Joomla.

- La même année que PHP, apparaît dans le milieu du numérique le langage JavaScript. Créé par Brendan Eich, JavaScript sert également dans la création de sites web. Il permet de gérer les docs PDF, les navigateurs, etc. Gmail, Adobe Photoshop ou encore Mozilla Firefox utilisent largement le langage JavaScript. Il fait également partie des langages informatiques les plus enseignés. JavaScript est aussi, depuis de nombreuses années, le langage informatique le plus populaire.

- En 2000, le langage C# apparaît chez Miscrosoft. Le but est d'aligner les caractéristiques de C++ avec celles de Visual Basic. Très proche du langage Java, le C# est très largement utilisé par Microsoft et notamment pour tous ses logiciels de bureautique [4].

1.3 Ordinateur

Pour traiter une information, un microprocesseur seul ne suffit pas, il faut l'insérer au sein d'un système minimum de traitement programmé de l'information. John Von Neumann est à l'origine d'un modèle de machine universelle de traitement programmé de l'information (1946). Cette architecture sert de base à la plupart des systèmes à microprocesseur actuel. Elle est composé des éléments suivants :

- une unité centrale.
- une mémoire principale.
- des interfaces d'entrées/sorties.

Les différents organes du système sont reliés par des voies de communication appelées bus.

1.3.1 L'unité centrale

Elle est composée par le microprocesseur qui est chargé d'interpréter et d'exécuter les instructions d'un programme, de lire ou de sauvegarder les résultats dans la mémoire et de communiquer avec les unités d'échange. Toutes les activités du microprocesseur sont cadencées par une horloge. On caractérise le microprocesseur par :

- sa fréquence d'horloge : en MHz ou GHz.
- le nombre d'instructions par secondes (en MIPS) qu'il est capable d'exécuter.
- la taille des données (en bits) qu'il est capable de traiter.

1.3.2 Le processeur

C'est le cerveau de l'ordinateur. Il réalise tous les calculs nécessaires au fonctionnement de l'ordinateur. C'est notamment la fréquence du processeur, c-à-d., la vitesse à laquelle il travaille, qui détermine la rapidité de votre ordinateur. Cette fréquence s'exprime en Giga Hertz (GHz). On dit les ordinateurs à **multicoeurs**

c-à-d., qu'ils possèdent plusieurs processeurs pour effectuer plus rapidement les tâches demandées. Ainsi :

- Dual Core = 2 coeurs;
- Quad Core = 4 coeurs;
- Hexa Core = 6 coeurs;
- Octa Core = 8 coeurs.

Il existe différents modèles de processeurs et cette technologie évolue rapidement. Les processeurs des marques AMD et Intel sont les plus fréquemment utilisés.

Dans la marque Intel par exemple, on voit généralement les modèles suivants :

- Intel Core i3 qui correspond à l'entrée de gamme des processeurs Intel Core;
- Intel Core i5 qui constitue le milieu de gamme;
- Intel Core i7 et Intel Core i9 qui sont de la gamme supérieure.

1.3.3 La mémoire centrale

Elle héberge les instructions du programme en cours d'exécution avec les données associées. Physiquement, elle se décompose souvent en :

- une mémoire morte (ROM = Read Only Memory) chargée de stocker le programme. C'est une mémoire à lecture seule.
- une mémoire vive (RAM = Random Access Memory) chargée de stocker les données intermédiaires ou les résultats de calculs. C'est la mémoire temporaire de l'ordinateur, c'est là que sont stockés tous les fichiers sur lesquels l'utilisateur est en train de travailler. On peut lire ou écrire des données dedans, ces données sont perdues à la mise hors tension (les informations sont supprimées lors de l'arrêt de l'ordinateur).
  Plus cette mémoire est importante, plus l'ordinateur travaille facilement et rapidement et plus il peut gérer des tâches différentes. La capacité de cette mémoire s'exprime en "Gigaoctets (Go)". La mémoire vive se présente sous forme de petites barrettes que l'on insère dans la carte mère. Un ordinateur peut ainsi comporter plusieurs barrettes pour avoir au total (cela dépend des ordinateurs) 4 - 8 - 16 ou 32 Go de RAM.

1.3.4 Les périphériques de stockage

On appelle "périphérique" tout matériel électronique pouvant être raccordé à un ordinateur. On peut distinguer les périphériques d'entrée et de sortie, les périphériques de stockage et d'autres. Les périphériques de stockage sont destinés pour stocker les données et sont considérés comme des mémoires secondaires. Toutes les données (photos, vidéos, musiques, ...) sont généralement stockées sur le disque dur. Dans certains cas, on peut avoir besoin de les copier sur un autre support. Par exemple, pour sauvegarder des données, pour transférer des données d'un ordinateur à un autre. Pour cela, vous pouvez par exemple utiliser un disque dur, une disquette, un disque dur externe, une carte mémoire, un DVD ou CDROM, une clé USB, etc, ...

Le disque dur C'est le support sur lequel on peut stocker des informations. Les capacités de stockage ne cessent d'augmenter et permettent donc d'enregistrer un grand nombre de données : documents, photos, films... Il y a actuellement deux types de disques durs : SSD et HDD. Les SSD ont l'avantage d'être extrêmement rapides, mais ils sont plus chers et de capacité limitée.

Le disque dur externe Un disque dur externe est un objet qui permet de stocker des données informatiques. Physiquement, il est embarqué dans un boîtier solide et qui se relie à l'ordinateur via un câble USB. Il peut être utilisé pour sauvegarder des données de l'ordinateur ou enregistrer des fichiers plus lourds (films) pour pouvoir les transporter.
Par rapport à une clé USB, le disque dur est généralement plus encombrant, mais il permet également de stocker plus de données. Les disques durs externes actuels ont généralement une capacité qui varie de 500 gigas à 5 téras (5000 gigas).

Les disques: CDROM, DVD et Blu-Ray Dans le passé, les disques étaient utilisés pour stocker des données informatiques et pour faire des sauvegardes. Mais depuis l'arrivée sur le marché de supports ayant une plus grande capacité de stockage, les disques sont de moins en moins utilisés. Les disques ont tous la même dimension (12 cm de diamètre), mais ils se différencient par leur capacité de stockage. Physiquement, on peut catégoriser les disques en trois types qui sont ainsi :

- 700 Mo pour le CD (utilisé pour les albums de musique);
- 4,7 Go pour le DVD (utilisé pour les films);
- 25 Go pour le Blu-Ray (utilisé pour les films en haute définition).

La carte mémoire Une carte mémoire est une petite carte qui permet de stocker des données dans un appareil. Ces cartes sont souvent utilisées dans des appareils photo, caméscopes, smartphones, tablettes. Il en existe de différents types : cartes SD, cartes miniSD, cartes microSD. Lorsque vous souhaitez acheter une carte mémoire, renseignez-vous pour savoir quelle carte est compatible avec votre appareil, c'est indiqué dans le mode d'emploi des appareils. Actuellement, ces cartes ont généralement une capacité allant de 16 Go à 128 Go.

La clé USB Une clé USB est un petit objet qui permet de stocker des données informatiques (films, photos, musiques, fichiers Word). Elle se connecte principalement à l'ordinateur, et aussi à certains appareils comme des chaines hifi, des autoradios, des téléviseurs, des lecteurs DVD à travers le port USB. Les clés USB actuelles ont généralement une capacité allant de 8 à 128 Go [3] [5] [8].

1.4 Processus de développement d'un programme

L'écriture d'un programme n'est qu'une étape dans le processus de programmation que le montre dans la figure 1.1.

1.4.1 Analyse et étude des besoins

Tout problème doit soumettre à une étape préalable q'on appelle l'analyse. Cette étape a pour but de spécifier les besoins de l'utilisateur et de détermminer les résultats attendus de la résolution du problème.

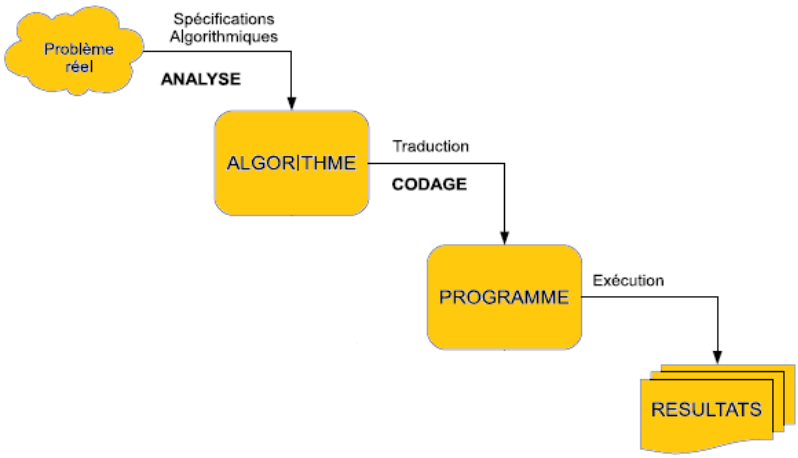


FIG. 1.1 – *Etapes du processus de programmation*

1.4.2 Etablissement d'un algorithme

Un algorithme décrit les étapes à suivre pour réaliser une ou plusieurs taches. Plus de détails, un algorithme est une suite d'actions, qui une fois conçu correctement conduit au codage de son programme [1].

Selon Larousse l'algorithme est un ensemble de règles opératoires dont l'enchaînement permet de résoudre un problème au moyen dŠun nombre fini d'opérations.

Structure générale d'un algorithme

La présentation d'un algorithme sous forme de blocs est très proche du langage de programmation. Sa structure est composée des éléments suivants:

```
Algorithme <identificateur_nom>;  {En-tête}
     <Constantes>
     :
     {Fonctions et/ou de procédures}    {Partie déclarations}
Début
     <partie actions ou traitement>;    {Corps de l'algorithme}
Fin.
```

Dans la suite, nous décrivons les fonctionalités de ces éléments.

En-tête Il s'agit de la première ligne de l'algorithme. La syntaxe de l'en-tëte est définie par le mot clé *Algorithhme*, suivi d'un identificateur et se termine par un point-virgule.

Syntaxe : Algorithme <identificateur_nom>;

Partie déclaration Cette partie contient la déclaration de tous les structures manipulées (principalement les types et les variables) par l'algorithme. Elle associe à chaque structure un nom (identificateur), un type et éventuellement une valeur pour les constantes. La zone de dèclaration comprend :

- Déclaration des étiquettes
- Déclaration des constantes avec CONST
- Déclaration des types avec TYPE
- Déclaration des variables avec VAR
- Déclaration des fonctions et/ou des procédures

Partie de traitement Cette partie représente le corps de l'algorithme et est constituée d'un ensemble d'actions appelées aussi instructions. Ces actions sont destinées à réaliser un traitement d'un problème donné. Une instruction est une phrase du langage représentant un ordre ou un ensemble d'ordres qui doivent être exécutés par la machine. L'ensemble d'instructions doit débuter par 'Début' et se termine par 'Fin'. Deux catégories d'instructions sont distingués [6]:

* Les instructions simples :
  - affectation, appels, branchement, ...
* Les instructions structurées :
  - instructions composées
  - instructions itératives
  - instructions conditionnelles

1.4.3 Codage ou traduction de l'algorithme

Un algorithme doit être exprimé dans un langage de programmation pour être compris et exécuté par la machine. Le programme constitue le codage d'un algorithme dans un langage de programmation donné, et qui peut être traité par une machine donnée.

1.4.4 Exécution du programme

Dans cette partie, l'exécution du programme est précédée par la phase de compilateion qui se réalise par un compilateur Pascal (par exemple, Turbo Pascal). Dans cette phase, le code source stocké dans le fichier .pas est vérifié pour des éventuels erreurs syntaxiques. Une fois les erreurs sont localisées et corrigées, un code objet est généré automatiquement dans un fichier .obj. Une deuxème phase d'édition des liens est indispensable dans l'environnement de Pascal afin de générer le fichier exécutable .exe. L'exécution réelle du programme est maintenant prète. Durant cette phase, on teste le programme avec un jeu de données d'entrée afin de valider le programme sur les deux plans : sémantique et conception.

Chapitre 2

Langage Pascal

Le langage de programmation Pascal a ètè conçu au début des années 70 par N. Wirth et s'est développé dans les universitès et la communauté scientifique. Puisque ce langagel est facile à enseigner et à apprendre, il est devenu un outil d'apprentissage et d'éducation puiissant. Il permet d'écrire des programmes trés lisibles et structurés et aussi dispose entre autres de facilités de manipulation de données [6].

2.1 Structure générale d'un programme Pascal

Un programme Pascal est composé des éléments suivants:

```
Program <identificateur>;  {l'entête du programme}
 {Partie déclarations}
     {Uses: les unités}
     {label: les étiquettes}
     {Const: les constantes}
     {Type: les types}
     {Var: les variables}
     {les procédures et les fonctions}

 Begin
 {Partie traitement}
     {suite d'actions ou instructions}
 End.
```

2.1.1 En-tête

Il s'agit de la première ligne d'un programme PASCAL. L'en-tête est définie par le mot clé PROGRAM suivi d'un identificateur, éventuellement suivi d'une liste de paramètres situés entre parenthèses. La ligne est terminée par un point-virgule.

Syntaxe : PROGRAM identificateur (id1,id2, ..., idn) ;

2.1.2 Partie déclaration

La zone de dèclaration comprend :

- Déclaration des unités avec USES
- Déclaration des étiquettes avec LABEL
- Déclaration des constantes avec CONST
- Déclaration des types avec TYPE
- Déclaration des variables avec VAR
- Déclaration des fonctions et/ou des procédures

[6]

2.1.3 Partie traitement

Une instruction est une phrase du langage représentant un ordre ou un ensemble d'ordres qui doivent être exécutés par la machine. La partie de traiteement débute les instructions par 'BEGIN' et se termine par 'END'. Dans cette partie, on distingue [6]:

* Les instructions simples :
  - affectation, appels, branchement, ...
* Les instructions structurées :
  - instructions composées
  - instructions itératives
  - instructions conditionnelles

2.2 Détails sur les déclarations

Avant de détailler cette partie, nous devons tout d'abord donner la signification de l'identificateur ou le nom attribué à tout objet employé dans la déclaration.

2.2.1 Identificateurs :

C'est un nom que l'on attribue à toute entité manipulée dans un algorithmme (ou programme). Les identificateurs sont donnés par l'utilisateur à son choix et selon certaines règles:

1. Le nom doit commencer par une lettre et non par un chiffre;
2. Il doit être constitué uniquement de lettres, de chiffres et du soulignement(les caractères de ponctuation et les espaces sont non acceptés);
3. Il doit être différent des mots clés réservés à l'algorithme ou au langage de programmation (par exemple: var, begin, sqrt, write . ..).

En Pascal, tout nouveau identificateur utilisé dans un programme doit être explicitement déclaré.

2.2.2 Déclarations de constantes

Une constante est désignée par un identificateur et une valeur, qui sont fixés en début de programme, suite au mot clé CONST. La valeur ne peut pas être modifiée, et ne peut pas être une expression.

Syntaxe identificateur = valeur_constante;
ou identificateur : type = valeur_constante;

Dans la première forme, le type est sous-entendu (si il y a un point, c'est un réel, sinon un entier ; si il y a des quotes, c'est un caractère (un seul) ou une chaîne de caractères (plusieurs) [7].

Remarque: L'utilisation de constantes en programmation est fortement conseillée.

2.2.3 Déclaration des types

a- Les types simples standards Appelés aussi types élémentaires, ces types sont à valeur scalaire. On peut distinguer les types standards tels que les types entiers, booléens, caractères et chaîne de caractères et les types non standards tels que les types énumérés et les intervalles.
La table ci-dessous récapitules les types de base standards qu'on utilise en Paacal :

TAB. 2.1 – *les types de base standards*

type	description	exemple
entier	un objet de type entier prend ses valeurs dans l'ensemble des entiers relatifs Z. En effet, un entier s'écrit comme suit: [+/-] <valeur>. Les signes +/- sont optionnels.	+205, -110, 1013 sont des valeurs entières valides.
réel	un objet de type réel prend ses valeurs dans l'ensemble des réels R. En effet, un réel s'écrit comme suit: [+/-] <partie entière>.<partie fractionnaire>E <entier>; E désigne la puissance de 10.	+34.06, -0.5 ≡ -5E-1, $-0.025 \times 10^3 \equiv$ -2.5E+1.
logique (ou booléen)	contient uniquement les deux valeurs vrai et faux.	Z ← X <Y. Z est de type logique, il reçoit vrai si l'expression X <Y est vraie, sinon faux.
caractère	un objet de type caractère prend ses valeurs dans l'ensemble des caractères alphabétiques minuscules et majuscules, numériques et caractères spéciaux ∗, + ,/,?, < , >, etc.	'b' <'c' <'e'.
chaîne de caractères	un objet de type chaîne de caractères est composé d'un ensemble d'objets de type caractère.	'8 MAI 45', 'Section', 'ST2023'.

Type entier : integer Entier signé en complément à deux sur 16 ou 32 bits, selon le syystème d'exploitation installé ssur la machine et le compilateur utilisé:

16 pour Turbo Pascal, 32 pour Delphi.
Sur 16 bits, à valeur dans -32768 ... +32767 ($-2^{15}... + 2^{15} - 1$).
Sur 32 bits, à valeur dans -2147483648 .... +2147483647 ($-2^{31}... + 2^{31} - 1$).

La table 2.2 ci-dessous montre quelques exemples d'opérations sur les entiers :

TAB. 2.2 – *Opérations sur les entiers*

abs(x)	valeur absolue de $\|x\|$
pred(x)	x - 1
succ(x)	x + 1
odd(x)	true si x est impair, false sinon
sqr(x)	le carré de x
+ x	identité
- x	signe opposé
x + y	addition
x - y	soustraction
x * y multiplication	
x / y	division, fournissant un résultat de type réel
x div y	dividende de la division entière de x par y
x mod y	reste de la division entière, avec y non nul.

Type réel : real Utilisable pour représenter les réels et les entiers élevés. La représentation binaire d'un réel commence à 32 bits, et dépend étroitement du système d'exploitation de la machine et aussi du compilateur.

Les fontions posssibles sont:

- abs(x), sqr(x), +x, -x. Si l'un au moins des 2 arguments est réel, le résultat est réel pour : x - y, x + y, x * y.abs(x), sqr(x), +x, -x. Si l'un au moins des 2 arguments est réel, le résultat est réel pour : x - y, x + y, x * y.
- Résultat réel que l'argument soit entier ou réel : x / y (y doit être non nul) ; fonctions sin(x), cos(x), exp(x), ln(x), sqrt(x) (square root, racine carrée).
- Fonctions prenant un argument réel et fournissant un résultat entier : trunc(x) (partie entière), round(x) (entier le plus proche). Si le résultat n'est pas représentable sur un integer, il y a débordement.Fonctions prenant un argument réel et fournissant un résultat entier : trunc(x) (partie entière), round(x) (entier le plus proche). Si le résultat n'est pas représentable sur un integer, il y a débordement.

Type booléen: boolean Un type défini par les deux valeurs logiques : true (vrai) et false (faux).

- Ce type permet la manipulation avec des opérateurs logiques : not (néégation), and (et), or (ou).
- Les opérateurs de comparaison : <, >, <=, >=, = , <>sont utilisés pour compareer deux variables:(entier-entier, réel-réel, entier-réel, caractèrre-caractère, booléen-booléen).

Le resultat d'une comparaison est un booléen.
On peut comparer 2 booléens entre eux, avec la relation d'ordre false <true.

- En codage de bits, les valeurs booléennes sont codées sur 1 bit, avec 0 pour false et 1 pour true.

Type caractère : char Un seul caractère entouré par deux apostrophes (' ').

- 256 caractères alphnumériques sont standardisés avec un ordre selon la table ASCII. Par exemple, l'ordre de 'A' est 65, et celui de 'a' est 97, etc.
- Les opérateurs sur les caractères sont :
  ord(c) numéro d'ordre dans le codage ASCII.
  chr(a) retourne le caractère dont le code ASCII est a.
  succ(c) donne le caractère qui suit c dans l'ordre ASCII.
  prec(c) donne le caractère qui précède c dans l'ordre ASCII.
- le blanc ' ' est un caractère dont son code ASCII est 32.

Type chaîne de caractères : string C'est un tableau de caractères. Ce type n'est pas disponible sur tous les compilateurs. Il est Possibile d'accéder à un caractère particulier de la chaîne, en indiçant sa position dans la variable de type string.

b- Les types simples non standards

Type intervalle C'est un sous-ensemble de valeurs consécutives d'un type hôte.

Syntaxe: variable : N..M;
où N et M sont des constantes du même type qui représentent respecctivement la borne inférieure et la borne supérieure de l'intervalle [N, M].

Exemple:

```
Var
    pourcentage : 0 .. 100; {type hote est integer}
    chiffre : '0' .. '9'; {type hote est char}
    valide : false .. true; {type hote est boolean}
```

Remarques:

* le type hôte soit codé sur un entier (signé ou non, sur un nombre de bits quelconque). On dit alors que ce type hôte est un type simple. Ainsi les types integer, char et boolean sont des types simples.
* Par contre 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 consécutifs >.
* Un autre cas de type non simple est le type string pour les chaînes de caractères, qui n'est pas codé sur un mais sur plusieurs entiers. On ne peut donc pas déclarer 'aaa'..'zzz'.

Type énuméré Il s'agit de plusieurs valeurs ou cas finis, que peut prendre une variable. Un type énuméré permet de donner un nom significatif à ces valeurs.

Exemple:

```
Var feux : (Rouge, Orange, Vert, Clignotant);
   Begin { ... }
   if feux = Rouge then writeln('Etat est arret')
   else if feux = Orange then writeln('Etat est possible')
       else if feux = Vert writeln('Etat est autorisé')
   End.
```

Remarques:

* le type énuméré est toujours codé sur un entier, et les constantes sont attribuées les valeurs 0, 1, 2, 3 (la première constante prend la valeur 0).
* Le type énuméré étant codé sur un entier, on peut utiliser les opérateurs pred, succ et ord.
* déclarer un type intervalle à partir d'un type énuméré est possible (exemple : Rouge..Clignotant).

c- Les types structurés On a vu dans les sections précédentes les types simples pré-déclarés qui sont boolean, integer, real et char. Les types structurés sont des nouveaux types non simples déclarés par l'utilisateur.

2.2.4 Déclaration de variables

Une variable représente un objet d'un certain type; cet objet est désigné par un identificateur. Toutes les variables doivent être déclarées après le mot clé VAR.

Syntaxe identificateur : type ;

On peut déclarer plusieurs variables de même type en même temps, en les séparant par des virgules. Au niveau de la déclaration, les variables ont une valeur indéterminée. On initialise les variables juste après le BEGIN (on ne peut pas le faire dans la déclaration) [7].

Remarque: On ne peut pas utiliser une variable non initialisée dans le programme.

2.2.5 Détails sur les instructions

Cette partie représente le corps de l'algorithme et est constituée d'un ensemble d'actions appelées aussi instructions. Ces actions sont destinées à réaliser un traitement d'un problème donné. Une instruction est une phrase du langage représentant un ordre ou un ensemble d'ordres qui doivent être exécutés par la machine. Dans cette partie l'ensemble d'instructions débute par 'BEGIN' et se termine par 'END'.

Deux catégories d'instructions sont distingués:

* Les instructions simples :
  - affectation, appels, branchement, ...
* Les instructions structurées :
  - instructions composées
  - instructions itératives
  - instructions conditionnelles

La base de ces instructions est l'expression.

Expressions et opérateurs

L'expreession en Pascal est similaire à celle donnée en algoriithmique. Une expression désigne une valeur, exprimée par la composition d'opérateurs appliqués à des opérandes, qui sont : des valeurs, des constantes, des variables, des appels de fonction ou des sous-expressions. Il existent trois types d'expressions [6].

La table 2.3 ci-dessous récapitule les types d'expression qu'on utilise dans le code Pascal:

TAB. 2.3 – *types d'expression*

type	description	exemple
Expression de base	appelée aussi élémentaire, représente les valeurs que peut prendre une constante ou une variable.	-210, 15, +450, Vrai, Faux, 'XX#'.
Expression arithmétique	combinaison de variables et/ou de constantes numériques (entier et réel) et des opérateurs arithmétiques. Une expression arithmétique donne un résultat numérique dont le type est entier ou réel.	a * 8, (a + 4) / (b-c), k * (a1+a2)/(r1+r2).
Expression logique	combinaison de variables et/ou de constantes de type logiquue et d'opérateurs logiques. Une expression logique donne un résultat booléen (vrai ou faux).	($X \leq Y$) et Z est une expression logique avec Z une variable booléenne.

Les opérateurs sont utilisés par l'expression pour aboutir au résuliat. Un opérateur relie deux opérandes pour produire un résultat. En Pascal, plusieurs types d'opérateurs sont adoptés. Opérateurs arithmétiques : $\& + , - , /, *$, div (division entière) et mod (reste de division entière). Opérateurs rationnels: $> , >= , < , <= , = , <>$. Opérateurs logiques: and (conjonction entre deux expressions booléennes), or (disjonction entre deux expressions booléennes), Not (négation d'une expression booléenne). La table 2.4 ci-dessous récapitule les types d'opérateurs qu'on utilise en Pascal.

Instructions simples

L'instruction simple est une opération élémentaire, c'est-à-dire, l'ordre le plus basique que peut exécuter l'UC de la machine. Principalement, elle comporte

TAB. 2.4 – *types d'opérateurs*

type	description	exemple
Opérateurs arithmétiques	Ils permettent le calcul de la valeur d'une expression arithmétique.	+, -, /, *, div (division entière) et mod (reste de division entière).
Opérateurs rationnels	ils permettent de comparer deux valeurs de même type (numérique ou caractère) en fournissant un résultat booléen (vrai ou faux).	>, >=, <, <=, =, <>.
Opérateurs logiques	ces opérateurs sont appliqués à des données de type booléen et renvoient un résultat de type booléen.	Et (conjonction entre deux expressions booléennes), Ou (disjonction entre deux expressions booléennes), Non (négation d'une expression booléenne).

deux éléments: l'action et les entités associées pour l'achevement de cette action. Par exemple, l'instruction d'addition a + b fera intervenir l'opérateur d'addition et les valeurs des deux opérandes a et b. Pour ce genre d'instructtions, nous distinguons: l'affectation et les opérations d'entrée et de sortie comme des instructions simples [1]. La table 2.5 ci-dessous récapitule les instructions d'affectation et d'opérations d'entrée et de sortie qu'on utilise dans l'algorithme :

TAB. 2.5 – *Instructions simples*

type	description	exemple
L'affectation	elle consiste à attribuer une valeur directe ou indirecte (sous forme d'expression qui nécéssite une pré-évaluation) à une variable. Cette valeur sera stockée dans une zone mémoire. L'affectation est réalisée au moyen de l'opérateur ←.	X ← 50: affecter la valeur constante 50 à la variable X.
L'opération d'entrée (lecture)	elle permet d'entrer des données à partir d'un périphérique d'entrée (clavier). Syntaxe: Lire(variable)	Lire(X), Lire(Y) ou bien Lire(X, Y): affecter des valeurs aux variables X et Y à partir du clavier.
L'opération de sortie (écriture)	elle permet d'afficher la valeur d'une variable ou d'une expression sur un périphérique de sortie (écran). Syntaxe: Écrire(expression):	Ecrire('Faculté ST'): afficher la chaine 'Faculté ST'.

Remarques :

1. Il est possible d'écrire: X ← Y ← Z+12, on affecte la valeur Z+12 à la variable Y, puis à la variable X la valeur de Y (résultat de Y ← Z+12). La valeur finale de X sera Z+12.
2. les instructions j ← j+1 et j←j-1 représentent respectivement l'incrémentation et la décrémentation.
3. On n'affecte jamais une valeur à une expression (l'expression est toujours à droite de l'affectation).

Instructions structurées

Notons qu'il existe d'autres types d'instructions, à savoir, l'instruction structurée (structure conditionnelle et itérative) et l'instruction composée qui peut contenir à la fois des instructions simples et structurées.

Premièrement, les instructions composées permettent de regrouper, dans un même bloc, un ensemble d'instructions qui seront exécutées au même niveau.

Syntaxe : Séquence de deux ou plusieurs instructions comprises entre BEGIN et END et séparées par des points virgules [7].

2.3 Les tableaux

Les variables simples ne sont pas suffisantes pour stocker les données pour certains problèmes complexes. Pour remédier à cette lacune, une alternative consiste à penser à l'utilisation d'une nouvelle structure de données indicées qui regroupe plusieurs valeurs de même type qu'on appelle les tableaux.

Formellement, un tableau d'ordre n, un vecteur de n éléments de même type disposés de n cases en mémoire. Un tableau possède un identificateur (nom du tableau) et un indice qui indique le numéro de la case.

2.3.1 Les tableaux unidimensionnels

Définitions et déclaration de type

Syntaxes

Syntaxe algorithmique En algorithmique, la déclration d'un tableau se fait avec deux syntaxes. La première syntaxe est la suivante:

Var <nom_tableau>: Tableau[<taille>] de <type_elements>;

Alors que, la seconde syntaxe est comme suit:

Type <type_tableau>= Tableau[<taille>] de <type_elements>;
Var <nom_tableau>: <type_tableau>;

sachant que
<type_tableau>: type déclaré par l'utilisateur défini une nouvelle structure du tableau.
<nom_tableau>: nom attribué au tableau.
<taille>: nombre d'éléments du tableau.
<type_elements>: type des éléments du tableau.

Exemple

```
Var T: Tableau[1..100] de caractères;
    V: Tableau['A'..'F'] de réels;
```

T est un tableau de 100 cases destinées à contenir des éléments de type caractère. V est un tableau de 6 cases (nombre de lettres comprises entre 'A' et 'F') destinées à contenir des éléments de type réel.

Syntaxe en langage Pascal En langage Pascal, on peut déclarer un tableau directement ou après la définition de type. La syntaxe est la suivante:

Var <nom_tableau>: array[I] of <type_elements>;

I étant un type intervalle, et <type_elements>un type quelconque.
Ce type définit un tableau comportant un certain nombre d'éléments de type <type_elements>, chaque élément est repéré par un indice de type I.

Exemple

```
Type vec_t = array [1..10] of integer;
Var v : vec_t;
```

v est un tableau de 10 entiers, indicés de 1 à 10.

2.3.2 Les tableaux multidimensionnels

On peut créer des tableaux à plusieurs dimensions de plusieurs manières :

- v1 : array [1..10] of array [1..20] of real
  $\longrightarrow$ tableau de 10 éléments, chaque élément étant un tableau de 20 réels.
  On accède à l'élément d'indice i dans 1..10 et j dans 1..20 par v1[i][j].
- v2 : array [1..10, 1..20] of real
  $\longrightarrow$ tableau de 10×20 réels.
  On accède à l'élément d'indice i dans 1..10 et j dans 1..20 par v2[i,j].

Exemple Mise à 0 du tableau v2.

```
Var
    v2 : array [1..10, 1..20] of real;
    i, j : integer;
Begin
    for i := 1 to 10 do
        for j := 1 to 20 do
        v2[i,j] := 0.0;
End.
```

Cet exemple de programme affiche le résultat suivant:

3	4	5
4	5	3
5	3	4
3	4	5

Remarques:

1. On déclare la procédure avant BEGIN du programme principal.
2. On peut appeler une procédure depuis une autre procédure, à condition que la procédure appelante soit déclarée après la procédure appelée.
3. On peut aussi appeler une procédure depuis elle-même; c'est ce qu'on appelle la récursivité.

2.3.3 Le type chaine de caractères

Une chaine de caractères est une suite ordonnée de caractères, elle est considérée comme un tableau de caractères. Chaque caractère se trouve à une position donnée dans la chaine qu'on appelle son rang dans la chaine. Le rang dans la chaine est équivalent à l'indice dans le tableau.
Un caractère peut être:

- une lettre ou un caractère alphabétique;
- un chiffre ou un caractère numèrique;
- un signe de ponctuation;
- un symbole ou un caractère spécial (*, _, $, #, /, ...).

En Pascal, la chaine typique est une séquence de caractères possèdant un compteur de taille à son origine. Chaque chaine possède une taille fixée par défaut à 255 [2].

Exemple

```
Var
  nom : string ; // chaine de 255 caractères
  titre : string[50]; // chaine de 50 caractères.
```

Les chaines Pascal sont limitées à 255 caractères. Une chaine est presque un tableau : premier_car := nom[1] ;

Syntaxes

Syntaxe en algorithmique Le type de chaine de caractères se présente sous la forme suivante:

Type <nom-chaine>= chaine[*longueur*];

où *longueur* indique la taille maximale d'une variable de ce type.

Syntaxe en langage Pascal On stocke une chaine de caractère telle que 'Hello' dans un objet de type string de la manière suivante :

Type <nom-chaine>= string[*longueur*];

où *longueur* est une constante entière donnant le nombre maximum de caractères pouvant être mémorisés. Pascal réserve un tableau [0..*longueur*-1] de caractères. Affecter une chaine plus longue que l'espace réservée à la déclaration est une erreur fatale.

Exemple

```
Var s : string[80];
Begin
    s := 'Le ciel est bleu.';
    writeln (s);
End.
```

Opérations sur les chaines

Les opérations sur les chaines de caractères sont utiles dans le code Pascal. On cite ici les principales:

1. Comparaison de deux chaînes: la comparaison s'effectue par comparer les caractères de même rang dans les deux chaînes.
   Les opérateurs utilisés sont =, <>, <, >, <=, >= et le résultat est un booléen.
   'baby' < 'sapin' car le code ASCII de 'b' est inférieur au code ASCII de 's'.

2. Concaténation de deux chaînes en une seule chaîne par juxtaposer les caractères de la deuxième chaîne après les caractères de la premièree chaîne.
   L'opérateur utilisé est + et le résultat est de type string.
   a := c + d avec c et d de types string ou char.

3. Chaine vide (longueur 0) : a := ''.

4. Recopie une chaine dans une autre chaîne : a := b.

5. Longueur courante de a : length(a).

Exemple Manipuler deux chaines de caractères s_1 et s_2.

```
CONST chaineplus= 'lire la documentation'; // chaines de caractères constante
VAR s1, s2 : string[100];
    i : integer;
```

```
BEGIN
    s1 := 'veuillez ';
    s2 := s1 + chaineplus; // concatenation de deux chaînes
    writeln ('s2 = ', s2);
    writeln ('Longueur courante de s2 : ', length(s2) );
    write ('rangs de caractère ''e'' dans s2 : ');
    for i := 1 to length(s2) do
        if s2[i] = 'e' then write(i, ' ');
END.
```

2.4 Les procédures et les fonctions

2.4.1 Les procédures

Une procédure est un sous-programme. L'utilisation des procédures permet de découper un programme en plusieurs modules. Chaque procédure réalise une ou plusieurs taches, que l'on peut appeler en tout endroit du programme. On peut ainsi rappeler le code de la procédure dans différents endroits dans le programme principal. Lorsqu'on implémente le programme en termes de procédures, on fait ce qu'on appelle une approchee descendante, c-à-d., on part du général au détail.

Principe

Il s'agit simplement de représenter un bloc d'instructions par un identificateur. Ensuite, l'appel de cet identificateur à divers endroits du programme provoque l'exécution à chaque fois de ce bloc d'instructions.

Remarque: Les procédures se déclarent et s'utilisent exactement comme les fonctions excepté le fait qu'elles ne renvoient aucune valeur et qu'elles, par conséquent, n'ont pas de type.

Syntaxes

La déclaration des procédures se fait à la fin de la partie déclaration.

Syntaxe algorithmique

```
Procédure nom_procédure : (paramètres: types) Début
    { ... corps de la procédure ... }
Fin;
```

ou

- *Procédure* est le mot clé introduisant la déclaration de la procédure;
- *nom_ procédure* est le nom de la procédure;
- *(paramètres: types)* est la déclaration du type des paramètres ou arguments de la procédure;
- *corps de la procédure* est l'ensemble des instructions de la procédure. Il est délimité par 'début' et 'fin';

Exemple Déclaration d'une procédure qui calcule le minimum de deux réels.

```
Procédure Minimum (a, b:réels):
var min : entier;
Début
    Si a < b alors min <--- a sinon min  <--- b;
    Ecrire('le minimum est ', min);
Fin.
```

Syntaxe en Pascal Procedure Nom_procédure[paramètres1: Types[;[var] paramètres2: Types])];

- *Nom_procédure* nom de la procédure;
- Le mot *var* (optionnel) n'est utilisé que pour les paramètres de sortie et d'entrée/sortie;
- Les identificateurs qui suivent le nom de la procédure constituent l'ensemble des paramètres formels avec leur type:
  - *paramètres1* groupe de paramètres (d'entrée) avec leur type et sans l'anticédent *var*.
  - *paramètres2* groupe de paramètres (d'entrée/sortie) avec *var*.

Exemple Procédure qui calcule le minimum de deux réels en pascal.

```
Procedure Minimum (a, b:real);
var min : integer;
Begin
    if a < b then min <--- a else min  <--- b;
    write('le minimum est ', min)
End;
```

Procédure sans paramètres

Exemple

```
program exemple;
var x, y, z, t : integer;
Procedure Permutation_circulaire;  {Déclaration de la procédure}
Begin { Corps de la procédure }
    t := x; x := y; y := z; z := t;
End;
Begin { Programme principal }
    x := 3; y := 4; z := 5;
    writeln (x, ' ', y, ' ', z);
    Permutation_circulaire; { 1er appel de la procédure }
    writeln (x, ' ', y, ' ', z);
    Permutation_circulaire; { 2eme appel de la procédure }
    writeln (x, ' ', y, ' ', z);
    Permutation_circulaire; { 3eme appel de la procédure }
    writeln (x, ' ', y, ' ', z);
End.
```

Cet exemple de programme affiche le résultat suivant:

3	4	5
4	5	3
5	3	4
3	4	5

Remarques:

1. On déclare la procédure avant BEGIN du programme principal.
2. On peut appeler une procédure depuis une autre procédure, à condition que la procédure appelante soit déclarée après la procédure appelée.
3. On peut aussi appeler une procédure depuis elle-même; c'est ce qu'on appelle la récursivité.

Appels Pascal permet de réaliser des appels de procédures d'une manière consécutive. A ce stade, on peut appeler une deuxième procédure depuis la première procédure (principle), à condition que la deuxième soit déclarée avant la première.

Exemple

```
program exemple;
var x, y, t : integer;
{Déclaration des procédures echanger et afficher}
Procedure echanger;
Begin { Corps de la procédure }
    t := x; x := y; y := t;
End;
Procedure afficher;
Begin { Corps de la procédure }
    writeln (x, ' ', y);
End;
Begin { Programme principal }
    x := 10; y := 4;
    afficher;
    echanger; { 1er appel de la procédure }
    afficher;
End.
```

Variables locales Les structures de données du programme qui ne sont utiles que dans la procédure peuvent se déclarer localement au niveau de la procédure.

Remarques:

- Une variable locale n'existe que pendant l'exécution de la procédure.
- Le programme principal n'accède plus à une variable locale d'une procédure.

– Une procédure de son coté n'accède jamais à une variable locale d'une autre procédure.

Exemple

```
program exemple;
var x, y : integer;
Procedure echanger;
Var t: integer; {variables locales}
Begin { Corps de la procédure }
    t := x;
    x := y;
    y := t;
End;
Procedure afficher;
Begin { Corps de la procédure }
    writeln (x, ' ', y);
End;
Begin
   {Corps du programme principal}
End.
```

Portée des variables Les variables déclarées dans le VAR du programme principal sont appelées des variables globales. Elles existent pendant toute la durée du programme et sont accessibles de partout. Une variable locale à une procédure, portant le même nom qu'une variable globale, masque la variable globale pendant l'exécution de cette procédure.

Exemple

```
program exemple;
var x : integer;
Procedure tester;
var x: integer; {variables locales}
Begin { Corps de la procédure }
    x := 8;
    write('X locale est ', x);
End;
Begin {Corps du programme principal}
x := 12;
writeln ('X globale est ', x);
tester;
writeln ('X globale est ', x);
End.
```

Cet exemple de programme affiche le résultat suivant:

X	*globale est*	12
X	*locale est*	8
X	*globale est*	12

Procédure avec paramètres

Principe L'appel de la procédure sans paramètres est peu lourd, puisque le programme principal communique avec la procédure à travers des variables de portée globale. Alors que la procédure avec paramètres emploie le passage des paramètres pour communiquer avec l'appelant. Deux sortes de paramètres sont utilisés: données d'entrée et résultats. Ensuite, l'appel avec des arguments d'appel provoque l'exécution avec des résultats qui dependent de ces paramètres ou arguments d'appel.

Nous prenons l'exemple de la procédure calcul_arithmetique. La solution consiste à déclarer des paramètres à la procédure :

```
program exemple;
var a, b: integer;
    c, d: real;

{Déclaration de la procédure calcul_arithmetique}

Procedure calcul_arithmetique(x, y : integer; var z: integer);
Begin { Corps de la procédure }
       z := (x + y) / (x * y);
End;

Begin { Programme principal }
    writeln ('Entrez deux entiers non nuls : a = ? et b = ? ');
    readln (a, b);
    calcul_arithmetique(a, b, c); {1er appel avec passage de paramètres}
    calcul_arithmetique(a+1, b-1, c); {2ième appel avec passage de paramètres}
    writeln ('c =  ', c, ' d = ', d);
End.
```

Cet exemple de programme affiche le résultat suivant:

Entrez deux entiers non nuls : a = ? *et* b = ?
5 4
$c = 0.45 \quad d = 0.5$

Remarques:

1. On déclare la procédure avant BEGIN du programme principal.
2. On peut appeler une procédure depuis une autre procédure, à condition que la procédure appelante soit déclarée après la procédure appelée.
3. On peut aussi appeler une procédure depuis elle-même; c'est ce qu'on appelle la récursivité.

Paramétrage Utiliser des paramètres est une solution élégante. On se concentre sur l'exemple suivant pour bien comprendre le principe.

```
program exemple;
var a, b, c, d : real;

Procedure somme(x,y : real; var z : real); { paramètres }
Begin
    z := x + y;
End;
Begin
     write ('a b ? ');
     readln (a, b);
     somme (a, b, c); { a + b }
     somme (a, -b, d); {a - b}
     writeln ('c = ', c, ' d = ', d);
End.
```

Passage de paramètres Il y a deux sorte de passage de paramètres : le passage par valeur et le passage par référence [7].

- Passage par valeur : à l'appel, le paramètre est une variable ou une expression. C'est la valeur qui est transmise, elle sert à initialiser la variable correspondante dans la procédure.
- Passage par référence : à l'appel, le paramètre est une variable uniquement (jamais une expression). C'est l'adresse mémoire (la référence) de la variable qui est transmise, non sa valeur. La variable utilisée dans la procédure est en fait la variable de l'appel, mais sous un autre nom (le programme se réfère à la valeur d'origine, même dans le code de la routine. Ceci permet à la procédure ou à la fonction de modifier la valeur du paramètre).
- C'est le mot-clé VAR qui tranche sur le type de passage: si le passage se fait par valeur (pas de VAR) ou par référence (présence du VAR).

2.4.2 Les fonctions

Principe Il s'agit simplement de représenter un bloc d'instructions par un identificateur. Ensuite, l'appel de cet identificateur à divers endroits du programme provoque l'exécution à chaque fois de ce bloc d'instructions [7].

Syntaxes La déclaration des fonctions se fait à la fin de la partie déclaration.

Syntaxe algorithmique

```
Fonction nom_fonction : (paramètres: types): type_resultat;
Début
    { ... corps de la fonction ... } {Résultat de la fonction, du type type_resultat}
    nom_fonction <--- expression;
Fin;
```

où

- *Fonction* est le mot clé introduisant la déclaration de la fonction;
- *nom_fonction* est le nom de la fonction;
- *(paramètres: types)* est la déclaration du type des paramètres ou arguments de la fonction;
- *type_resultat* est le type du résultat fourni par la fonction;
- *corps de la fonction* est l'ensemble des instructions de la fonction. Il est délimité par 'début' et 'fin';
- *nom_fonction* ← *expression* est le renvoi du résultat qui se fait par l'intermédiaire du nom de la fonction via l'opérateur d'affectation.

Exemple Fonction équivalente qui traite l'exemple précédent de la procédure calcul_arithmetique.

```
Fonction calcul_arithmetique(x, y:entiers):entier
Début
       calcul_arithmetique <--- (x + y) / (x * y);
Fin.
```

Syntaxe en Pascal

```
Function nom_fonction ([var] paramètres: types): type_resultat;
Begin
    { ... corps de la fonction ... } {Résultat de la fonction, du type type_resultat}
    nom_fonction := expression;
End;
```

Exemple Nous reprenons l'exemple précédent de la procédure calcul_arithmetique et nous essayons d'employer une fonction à sa place qui fait le même traitement :

```
program exemple; var a, b: integer; c, d: real; {Déclaration de la
procédure calcul\_arithmetique}

Function calcul_arithmetique(x, y : integer): integer);
Begin {Corps de la fonction}
    calcul_arithmetique := (x + y) / (x * y);
End;
Begin { Programme principal }
    writeln ('Entrez deux entiers non nuls : a = ? et b = ? '); readln (a, b);
    c := calcul_arithmetique(a, b); {1er appel avec passage de paramètres}
    d := calcul_arithmetique(a+2, b-2); {2ième appel avec passage de paramètres}
    writeln ('c =  ', c, ' d = ', d);
End.
```

Cet exemple de programme affiche le résultat suivant:

Entrez deux entiers non nuls : a = ? et b = ?
25 4
$c = 0.29 \quad d = 13.5$

Remarques:

1. On déclare la procédure avant BEGIN du programme principal.
2. On peut appeler une procédure depuis une autre procédure, à condition que la procédure appelante soit déclarée après la procédure appelée.
3. On peut aussi appeler une procédure depuis elle-même; c'est ce qu'on appelle la récursivité.

Variables locales Les variables locales ont pour role de stocker les données temporairement jusqu'à la fin de traitement réalisé par la fonction tutelle de ces variables.

Exemple on fait ce qu'on veut du variable local *res* dans la fonction, et à la fin de la fonction on écrit nom_fonction := res;

```
Function nom_fonction : type_resultat; Var res : type_resultat;
Begin
    { ... dans le corps, on fait ce qu'on veut de res ...}
    nom_fonction := res;
End;
```

2.5 Les enregistrements et les fichiers

2.5.1 Les enregistrements

Type enregistrement

Il s'agit simplement de regrouper des proprietés sous forme de champs c_1, c_2, ..., c_n de différents types t_1, t_2, ..., t_n dans un objet composé de n proprietés.

Syntaxe

type_enregistrement = Record
c_1 : t_1;
c_2 : t_2;
⋮
c_n : t_n;
End;

Soit r une variable de ce type; on accéde aux champs de r par $r.c_1$, $r.c_2$, ...
Nous prenons un exemple d'enregistrement qui décrit le portrait d'une personne.

```
{ ... }
type personne_t = Record
    taille : real;
    cheveux : string;
    yeux : string;
end;
var amine, susane : personne_t;
```

Remarques:

1. Soient r_1 et r_2 deux variables de ce type d'enregistrement, pour recopier le contenu de r_2 dans r_1, il suffit d'écrire : $r_2 := r_1$;
2. Lorsqu'on crée un type t_2 à partir d'un type t_1, ce dernier doit être déclaré avant.
3. ...

Tableaux d'enregistrements

On peut créer des tableaux d'enregistrements, et des enregistrements qui contiennent des tableaux.

```
{ ... }
const MaxEleves = 35; MaxNotes = 10;
type
    note_t = array[1..MaxNotes] of real;
    eleve_t = Record
            age, nb_notes : integer;
            notes : note_t;
            moyenne : real;
    End;
    classe_t = array[1..MaxEleves] of eleve_t;
var c : classe_t;
```

2.5.2 Les fichiers

Les entrées/sorties dans un programme Pascal sont des échanges d'informations entre la mémoire de l'ordinateur et ses périphériques (disques, clavier, écran, imprimante, etc). Les opérations d'entrées/sorties se réalisent par le biais de fichiers séquentiels. Un fichier en mode séquentiel est une collection de données de même type (souvent de caractères), où les données sont lues ou écrites les unes après les autres, en commençant par le l'entète et on avance sans retour possible en arrière [7].

Remarques:

1. Un fichier peut être vide ;
2. il peut avoir une fin ou non ;
3. il peut être ouvert (accessible) ou fermé.

Notions générales

Sur un disque, un fichier a un chemin qui inclut son nom, par exemple 'c:\st\tp1.pas'

On peut coder ce chemin dans un string, par exemple $cheminf$. Dans un programme qui doit manipuler ce fichier, il faut une variable pour le désigner, par exemple f [7].

Opérations sur les fichiers a) Déclarer la variable f
f : text ; ou f : file of qqchose ;
b) Assigner la variable f au fichier de chemin *cheminf*
assign(f, nomf) ;
c) Ouvrir le fichier f pour pouvoir y lire ou y écrire les données
reset (f) ; ou rewrite (f) ;
d) Lire ou écrire des données
read (f, donnee) ; ou write (f, donnee) ;
e) fermeture le fihier
close (f) ;

Modes d'accès au fichier Il existe deux modes d'accès au fichier. On ouvre un fichier soit en mode lecture, soit en mode écriture. On ne peut pas faire les deux en même temps.

- En lecture : on fait reset(f); puis des read(f, ...). Si le fichier n'existe pas, il y a une erreur.
- En écriture : on fait rewrite(f); puis des write(f, ...). Si le fichier n'existe pas, un rewrite le crée. S'il existe déja, le rewrite l'écrase, c'est-à-dire que l'ancien contenu est définitivement perdu.

Les types de fichiers

On distingue deux types de fichiers: les fichiers de texte et les fichiers d'éléments [7].

Fichiers de texte Les fichiers de textes sont les fichiers que vous pouvez éditer, comme par exemple vos fichiers Pascal.

1. Déclaration
```
Var
    f : text;
    c : char;
    s : string[255];
    x : integer;
    r : real;
```

2. Lecture
   read (f, c); lit un caractère dans f.
   readln (f, s); lit une ligne complète dans f (tjrs readln sur un string).
   read (f, x); lit un entier où un réel dans f.

3. *É*criture
   write (f, c); écrit un caractère dans f.
   write (f, s); écrit une chaine ou string dans f.
   Pour passer à la ligne on fait writeln(f);
   write (f, x, ' '); écrit un entier, un réel ou un booléen dans f.
   write(f, r:8:2); écrit un réel formatté dans f.

Fichiers d'éléments Les fichiers d'éléments sont des copies de la mémoire vive, les éléments étant tous du même type. Le type d'élément peut être un type simple, un enregistrement, un tableau, etc.

1. Déclaration

```
Type element_t = record
                    age : integer;
                    majeur : boolean;
                 end;
Var
    f : file of element_t;
    e : element_t;
```

2. Lecture
   read (f, e); lit un élément dans f. On ne fait jamais de readln.

3. Écriture
   write(f, e); écrit un élément dans f. On ne fait jamais de writeln.

4. Exemple
   Mettre un vecteur vec de nb éléments dans un fichier.

```
{...}
Const nomf := 'c:\st\data.dat';
Var
    vec : array [1..vmax] of element_t;
    nb, i : integer;
    f : file of element_t;
Begin
    assign(f, nomf);
    rewrite (f);
    for i := 1 to nb do
        write (f, vec[i]);
   close (f);
End.
```

Chapitre 3

Series d'exercices de TP

Serie TP No1: Les tableaux

Cette série comporte des exercices avec solutions sur les tableaux (ou vecteurs) 1- et 2-dimensions. Les exercices vont traiter les différents problèmes et on se concentre sur la déclaration du tableau et les instructions pour parcourir ses éléments.

3.1 Les tableaux

3.1.1 Tableaux 1D

Exercice 01 Soit un tableau T (ou vecteur) à une dimension contenant n nombres entiers ($n \leq 100$), introduits par l'utilisateur. Ecrire des programmes Pascal qui permettent de calculer

a- le minimum, le maximum et la moyenne des éléments de T.
b- le produit de tous les éléments de T ainsi que le nombre de valeurs strictement positives.

et aussi de

c- déterminer les positions de l'apparition d'une valeur dans T.
d- inverser le contenu de T.
e- supprimer toutes les valeurs nulles dans T (la suppression d'un élémént T[i] provoque le décalage des autres éléments d'indice >= (i+1) à la position i).
f- mettre les valeurs négatives au début et les valeurs positives à la fin en utilisant un seul tableau.

Solution Ex 01-a
{minimum, maximum et moyenne des éléments d'un tableau de taille $n \leq 100$}

```
program minimum_maximum_moyenne;
var i, n, max, min,
somme:integer; moyenne:real;
    T : array [1..100] of integer;
```

```
Begin
    write('Donner la taille du tableau: ');
    readln(n);
    for i:=1 to n do
    begin
        write ('T[',i,']: ');
        readln (T[i]);
    end;
    max := T[1]; min := T[1]; somme := T[1];
    for i:=2 to n do
    begin
        somme := somme + T[i];
        if T[i] > max then max := T[i];
        if T[i] < min then min := T[i];
    end;
    moyenne := somme/n;
    writeln('Le minimum est ',min);
    writeln('Le maximum est ',max);
    writeln('Le moyenne est ',moyenne:2:2);
End.
```

Solution Ex 01-b

{produit de tous les éléments de T ainsi que le nombre de valeurs strictement positives}

```
program produit_tous_éléments;
var i,n,produit,nb_pos:integer;
    T : array [1..100] of integer;

Begin
    write('Donner la taille du tableau: ');
    readln(n);
    for i:=1 to n do
    begin
        write ('T[',i,']: ');
        readln (T[i]);
    end;
    produit := 1; nb_pos := 0;
    for i:= 1 to n do
    begin
        produit := produit * T[i];
        if T[i] > 0 then nb_pos := nb_pos + 1;
    end;
    writeln('Le produit est: ', produit);
    writeln('Le nombre des valeurs positives est: ', nb_pos);
End.
```

Solution Ex 01-c

{les positions de l'apparition d'une valeur dans un tableau T}

```
program apparition_valeur;
var i , n, nb, valeur : integer;
    T : array [1..100] of integer;
Begin
    write('Donner la taille du tableau: ');
    readln(n);
    for i:=1 to n do
    begin
        write ('T[',i,']: '); readln (T[i]);
    end;
    write('Donner la valeur à trouver: ');
    readln(valeur);
    nb := 0;
    for i:= 1 to n do
    begin
        if T[i] = valeur then
        begin
            writeln('position = ',i);
            nb := nb + 1;
        end;
    end;
    if nb = 0 then writeln('La valeur ', valeur, ' n''existe pas dans le tableau !');
End.
```

Solution Ex 01-d

{inversement du contenu d'un tableau T}

```
program inversement_tableau; var i , n, p : integer;
    T : array [1..100] of integer;

Begin
     write('Donner la taille du tableau: ');
     readln(n);
     for i:=1 to n do
     begin
         write ('T[',i,']= ');
         readln (T[i]);
     end;
     for i:= 1 to n div 2 do
     begin
         p := T[i];
         T[i] := T[n-i+1];
         T[n-i+1] := p;
     end;
     writeln('Voici le tableau inversé: ');
     for i:= 1 to n do writeln('T[',i,']= ',T[i]);
End.
```

Solution Ex 01-e

{suppression de toutes les valeurs nulles d'un tableau T}

```
program suppression_valeurs_nulles;
var i, j, n : integer;
    T : array [1..100] of integer;

Begin
    write('Donner la taille du tableau: ');
    readln(n);
    for i:=1 to n do
    begin
        write ('T[',i,']= ');
        readln (T[i]);
    end;
    {à chaque suppression de T[i], on décale tous les éléments d'indice >= (i+1)
    à la position i}
    i := 1;
    while i<=n do if T[i] = 0 then
                  begin
                        for j := i to n-1 do T[j] := T[j+1];
                        n := n -1;
                  end
                  else i := i +1;
    writeln('Voici le tableau après la suppression des valeurs nulles: ');
    for i:= 1 to n do writeln('T[',i,']= ',T[i]);
End.
```

Solution Ex 01-f

{mise les valeurs négatives au début et les valeurs positives à la fin en utilisant un seul tableau}

```
program mise_valeurs_négatives_positives;
var i, j, n, tmp:integer;
    T : array [1..100] of integer;

Begin
    write('Donner la taille du tableau: ');
    readln(n);

    for i:=1 to n do
    begin
        write ('T[',i,']= ');
        readln (T[i]);
    end;

    j := 1;
    while ((j<=n) and (T[j]<0)) do j := j + 1;
```

```
    i := j;
    while (j<=n) do
    begin
       if (T[j]<0) then
            begin
                 tmp  := T[i];
                 T[i] := T[j];
                 T[j] := tmp;
                 i := i + 1;
            end
      else j := j + 1;
    end;

    writeln('Le tableau après repositionnement des valeurs négatives et positives:');
    for i:= 1 to n do
        writeln('T[',i,']= ',T[i]);
End.
```

Exercice 02 Soient deux tableaux (ou vecteurs) T_1 et T_2 à une dimension contenant chacun n nombres entiers ($n \leq 100$), introduits par l'utilisateur. Ecrire un programme Pascal qui permet d'additionner les éléments de T_1 avec ceux de T_2. Le résulat doit être dans T_1.

Solution Ex 02

{addition les éléments des deux tableaux T1 et T2}

```
program addition_deux_tableaux;
var i , n, somme, produit : integer;
    T1, T2 : array [1..100] of integer;

Begin
    write('Donner la taille du tableau: ');
    readln(n);
    for i:=1 to n do
    begin
        write ('T1[',i,'] : '); readln (T1[i]);
    end;
    for i:=1 to n do
    begin
        write ('T2[',i,'] : '); readln (T2[i]);
    end;
    {addition élément par élément}
    for i:= 1 to n do
    begin
    T1[i] := T1[i] + T2[i];
    writeln('T1[', i, '] = ', T1[i] );
    end;
End.
```

Exercice 03 Ecrire un programme Pascal qui demande *á* l'utilisateur d'entrer 10 valeurs réelles et les stocke dans un tableau, puis le programme calcule et affiche la valeur
- minimale
- maximale
avec son rang.

Solution Ex 03
{calcul les valeurs: minimale et maximale avec leur rang}

```
program valeurs_minimale_maximale;
const n = 10;
type tab = array[1..n] of real;
var i, rmin, rmax: integer;
    min, max: real;
    T: tab;

Begin
    writeln ('Entrez les 10 valeurs réelles du tableau');
    for i := 1 to n do readln(T[i]);

    min := T[1]; max := T[1];
    rmin := 1; rmax := 1;
    for i := 2 to n do
        if min > T[i] then
        begin
             min := T[i];
             rmin := i;
        end
        else
        if max < T[i] then
        begin
             max := T[i];
             rmax := i;
        end;
    write('La valeur minimale est ', min, ');
    writeln (' de rang = ', rmin);
    write('La valeur maximale est ', max, ');
    writeln (' de rang = ', rmax);
End.
```

Exercice 04 Soient deux tableaux (ou vecteurs) U et V de taille n ($n = 3$) de valeurs entières introduites par l'utilisateur. Ecrire les programmes Pascal qui permettent de calculer le produit
a- scalaire $U.V = \sum_{x=1}^{n} U_x * V_x$
b- vectoriel $U \times V = (U_y.V_z - U_z.V_y, U_z.V_x - U_x.V_z, U_x.V_y - U_y.V_x)$
de U et V.

Solution Ex 04-a

{calcul le produit scalaire de deux vecteurs U et V.}

```
program produit_scalaire;
const n=3;
var i, produit : integer;
    U, V : array [1..n] of integer;
Begin
    writeln('Entrez le vecteur U: ');
    for i:=1 to n do
    begin
        write ('U[',i,'] de U: '); readln (U[i]);
    end;
    writeln('Entrez le vecteur V: ');
    for i:=1 to n do
    begin
        write ('V[',i,'] de V: '); readln (V[i]);
    end;
    writeln('calcul le produit scalaire:');
    produit := 0;
    for i:= 1 to n do  produit := produit + U[i]*V[i];
    writeln('Le produit scalaire de U et V est: ', produit);
End.
```

Solution Ex 04-b

{calcul le produit vectoriel $U \times V = (U_y.V_z - U_z.V_y, U_z.V_x - U_x.V_z, U_x.V_y - U_y.V_x)$.}

```
program produit_vectoriel;
const n=3;
var i: integer;
    U, V, W : array [1..n] of integer;
Begin
    writeln('Entrez le vecteur U: ');
    for i:=1 to n do
    begin
        write ('U[',i,'] de U: '); readln (U[i]);
    end;
    writeln('Entrez le vecteur V: ');
    for i:=1 to n do
    begin
        write ('V[',i,'] de V: '); readln (V[i]);
    end;
    writeln('calcul le vecteur W:');
    W[1] := U[2] * V[3] - U[3] * V[2];
    W[2] := U[3] * V[1] - U[1] * V[3];
    W[3] := U[1] * V[2] - U[2] * V[1];
    for i:= 1 to n do  writeln('W[',i,'] = ', W[i]);
End.
```

3.1.2 Tableaux 2D

Exercice 05 Ecrire un programme Pascal qui calcule et affiche l'addition de deux matrices des entiers M_1 et M_2 de taille n $\times$ n (n=10) et qui sont remplies par l'utilisateur. Le résultat de l'addition doit être dans M_1.

Solution Ex 05
{addition deux matrices}

```
program addition_deux_matrices;

const n = 10;
type mat = array[1..n, 1..n] of integer;
var i, j: integer;
    M1, M2: mat;

Begin
    writeln ('Entrez les nxn valeurs de la première matrice');

    for i := 1 to n do
        for j := 1 to n do
                readln(M1[i, j]);

    writeln ('Entrez les nxn valeurs de la deuxième matrice');
    for i := 1 to n do
        for j := 1 to n do
                readln(M2[i, j]);

    {M1 est aussi la matrice résultat de l'addition}
    for i := 1 to n do
    begin
        for j := 1 to n do
        begin
            M1[i, j] := M1[i, j] + M2[i, j];
            write(M1[i, j], ' ');
        end;
        writeln('\n');  {saut de ligne}
    end;
End.
```

Exercice 06 Ecrire un programme Pascal qui demande *á* l'utilisateur de remplir une matrice carrée de n $\times$ n entiers (n=10), puis le programme calcule et affiche la somme des valeurs de sa diagonale.

Solution Ex 06
{calcul la somme des valeurs de la diagonale}

```
program somme_valeurs_diagonale;
const n = 10;
type mat = array[1..n, 1..n] of integer;
var i, j, s: integer;
    M: mat;

Begin
    writeln ('Entrez les nxn valeurs de la matrice');
    for i := 1 to n do
        for j := 1 to n do
            readln(M[i, j]);

    s := 0;
    for i := 1 to n do
        for j := 1 to n do
            if i = j then
                s := s + M[i, j];

    writeln ('somme des valeurs du diagonal est ', s);

End.
```

Exercice 07 Une entreprise possède dans son effectif n employés (n un entier positif non nul introduit par l'utilisateur). Ecrire un programme Pascal qui permet d'entrer des salaires de tout les employés pour les 12 mois de l'année courante dans une matrice. Le programme doit déterminer et afficher le rang des employés (1 à n) et le mois (rang de 1 à 12) ayant un salaire
a- minimal.
b- maximal.

Solution Ex 07
{détermination et affichage des employés et le mois ayant le salaire (minimal, maximal)}

```
program employés_salaire_minimal_maximal;
const m=12;
type mat = array[1..100, 1..12] of real;
var i, j: integer;
    sal, min, max: real;
    M: mat;

Begin
    writeln('Entrez le nombre d''employés n');
    readln(n);
```

```
    if(n >= 1) then
    begin
        {remplir la matrice des salaires}
        for i:=1 to n do
            for j:=1 to m do  read(M[i, j]);

        {l'employé et le mois pour un salaire minimal}
        min_sal := M[1, 1]; {initialisation pour le minimum}
        min_rang := 1;
        min_mois := 1;
        {l'employé et le mois pour un salaire maximal}
        max_sal := M[1, 1]; {initialisation pour le maximum}
        max_rang := 1;
        max_mois := 1;
        for i:=1 to n do
            for j:=1 to m do
            begin
                if(min_sal > M[i, j])
                begin
                    min_sal := M[i, j];
                    min_rang := i;
                    min_mois := j;
                end
                else if(max_sal < M[i, j])
                begin
                    max_sal := M[i, j];
                    max_rang := i;
                    max_mois := j;
                end
            end;
        writeln('le salaire minimal est ', min_sal, ' pour l''employé ', min_rang,
        ' au mois ', min_mois);
        writeln('le salaire maximal est ', max_sal, ' pour l''employé ', max_rang,
        ' au mois ', max_mois);
    end
    else writeln('Erreur: valeur de n est négative ou nulle !');
End.
```

3.2 Les chaines de caractères

Exercice 08 Ecrire un programme Pascal qui permet de concaténer deux chaines de caractères constantes 'veuillez ' et 'lire la doc', puis le programme doit afficher
a- la chaine résulat avec sa longueur.
b- les positions de l'apparition du caractère 'l' dans la chaine résulat.

Solution Ex 08
{concaténation de deux chaines constantes puis les positions de l'apparition du caractère 'l' dans la chaine résultat.}

```
program concatenation_chaines_constantes;
const Initial = 'veuillez ';
      Slogan = 'lire la doc';
var s1, s2 : string[100]; i : integer;

Begin
    s1 := Initial;
    s2 := s1 + Slogan;
    writeln ('s2 = ''', s2, '''');

    writeln ('Longueur courante de s2 : ', length(s2));
    write ('Indices des ''l'' dans s2 : ');
    for i := 1 to length(s2) do
        if s2[i] = 'l' then
        write(i,' ');
    writeln;
End.
```

Exercice 09 Ecrire un programme Pascal qui permet de saisir deux chaines de caractères s1 et s2. Le programme doit permettre de
a- concaténer les deux chaines et le résultat sera dans s2.
b- inverser la chaine résultat s2.

Solution Ex 09
{concaténation de deux chaines saisies puis inverser la chaine résultat.}

```
program concatenation_chaines_saisies;

var s1, s2 : string[100];
    i, n : integer;
    c : char;

Begin
    writeln ('Entrez la première chaine s1 : ');
    readln(s1);

    writeln ('Entrez la deuxième chaine s2 : ');
    readln(s2);

    s2 := s1 + s2;
    writeln ('s2 = ''', s2, '''');
    writeln ('Longueur courante de s2 : ', length(s2) );
```

```
    {inverser la chaine s2}
    n := length(s2);
    for i := 1 to div(n, 2) do
    begin
        c := s2[n-i+1];
        s2[n-i+1]; := s2[i];
        s2[i] := c;
    end;
    {afficher la nouvelle chaine s2}
    writeln ('s2 = ''', s2, '''');
End.
```

Exercice 10 Ecrire un programme Pascal qui concatène une chaine de caractères constante 'ST' avec plusieurs caractères introduits par l'utilisateur et affiche la chaine de résultat sur écran. L'introduction des caractères se termine dès qu'on tape le caractère '.'.

Solution Ex 10

{concatène une chaine de caractères constante avec des caractères saisis en séquence.}

```
program concatenation_caractères_séquence;
var c : char;
    s : string[100];

Begin
    s := 'ST'; {initialiser s par la chaine 'ST'}
    c := ' '; {initialiser c par le caractère blanc}
    while c != '.' do
    begin
        writeln ('Entrer un nouveau caractère');
        readln(c);
        s := s + c;
    end;
    writeln ('la nouvelle chaine après concaténation est ', s);
End.
```

Serie TP No2: Les procédures et les fonctions

Cette série contient des exercices avec solutions sur les procédures et les fonctions avec ou sans paramètres. Des exercices de cette série sont proposés avec des solutions adaptées dans un nouveau contexte.

3.3 les procédures et les fonctions sans paramètres

Dans cette section, nous présentons les procédures et les fonctions sans paramètres. A noter que l'absence des paramètres dans ces routines nécessite d'employer les variables déclarées globalement pour certains traitements comme le retour des résultats. Dans la sous-section suivante, nous présentons des exercices qui emploient les procédures sans paramètres.

3.3.1 Les procédures

Exercice 01 Pour un entier positif ($n >= 1$), introduit par l'utilisateur. Ecrire deux programmes Pascal avec procédures pour calculer les sommes suivantes:
a- $S = 1 + 2 + ... + n$.
b- $S = \sum_{i=1}^{n} \sqrt{2i + 1}$.

Solution Ex 01-a
{trois procédures pour calculer la somme 1+2+....+ n}

```
program somme_nombres;

var i, n, s: integer;

{Déclaration de la procédure somme_for_1_n}
procedure somme_for_1_n;
begin
   s := 0; {initialisation de la somme s}
```

```
   for i:=1 to n do s := s+i;
end;
{Déclaration de la procédure somme_while_1_n}
procedure somme_while_1_n;
begin
   s := 0; {initialisation de la somme s}
   i:=1;  {initialisation du compteur i}
   while i<=n do begin
          s := s+i;
          i := i+1; {incrémentation du compteur i}
   end;
end;
{Déclaration de la procédure somme_repeat_1_n}
procedure somme_repeat_1_n;
begin
   s := 0; {initialisation de la somme s}
   i:=1;  {initialisation du compteur i}
   repeat
          s := s+i;
          i := i+1; {incrémentation du compteur i}
   until i>n;
end;

begin
     writeln('Entrez la valeur de n >= 1');
     readln(n);
     somme_for_1_n; {somme1: s=1+2+3+....+n avec FOR}
     somme_while_1_n; {ou somme2: s=1+2+3+....+n avec WHILE}
     somme_repeat_1_n; {ou somme3: s=1+2+3+....+n avec REPEAT}
     writeln('s = ', s);
end.
```

Solution Ex 01-b

{trois procédures pour calculer la somme $S = \sum_{i=1}^{n} \sqrt{2i+1}$ / $n \geq 1$}

```
program somme_racines;
var i, s, n:integer;

{Déclaration de la procédure somme_for_racines}
 procedure somme_for_racines;
 begin
    s := 0; {initialisation de la somme s}
    for i:=1 to n do s := s+sqr((2*i)+1);
 end;

 {Déclaration de la procédure somme_while_racines}
 procedure somme_while_racines;
 begin
    s := 0; {initialisation de la somme s}
```

```
    i:=1;  {initialisation du compteur i}
    while i<=n do
    begin
           s := s+sqr((2*i)+1);
           i := i+1; {incrémentation du compteur i}
    end;
 end;
 {Déclaration de la procédure somme_repeat_racines}
 procedure somme_repeat_racines;
 begin
    s := 0; {initialisation de la somme s}
    i:=1;  {initialisation du compteur i}
    repeat
           s := s+sqr((2*i)+1);
           i := i+1; {incrémentation du compteur i}
    until i>n;
 end;

Begin
      writeln('Entrez la valeur de n >= 1');
      readln(n);
      {somme1 avec FOR}
      somme_for_racines;

      {ou somme2 avec WHILE}
      somme_while_racines;

      {ou somme3 avec REPEAT}
      somme_repeat_racines;
      writeln('s = ', s);
End.
```

Exercice 02 Ecrire une procédure dans un programme Pascal qui permet de lire entier, un caractère, un boolean, un réel et une chaine de caractères et d'afficher cette séquence sur l'écran.

Solution Ex 02

{procédure afficher sans paramètres qui emploie les variables globales}

```
program afficher_valeurs;
var e : integer;
    c : char;
    b : boolean;
    r : real;
    s : string[32];

procedure afficher;
begin
    writeln(('entrez un entier, un caractère, un booléan, un réel puis une chaine');
```

```
    read(e, c, b, r, s);
    writeln (e, '|', c, '|', b, '|', r, '|', s);
end;

Begin
    afficher;
End.
```

Exercice 03 Ecrire une fonction dans un programme Pascal qui demande *á* l'utilisateur de calculer et afficher la somme donnée par l'expression suivante : $S = p + \left[\frac{\prod_{i=1}^{n} i}{\sum_{j=1}^{m} j}\right]$ avec $m \geq 1$, $n \geq 1$ et p des entiers entrés par l'utilisateur.

Solution Ex 03

{fonction sans paramètres de calcul d'une expression réelle qui emploie les variables globales}

```
program somme_reels;
var n, m, p, sum, prod, i, j : integer;
    s: real;
function calcul_expression: real;
begin
    {étape1: calcul le produit}
    prod := 1;
    for i := 1 to n do prod := prod * i;

    {étape2: calcul la somme}
    sum := 0;
    for j := 1 to m do sum := sum + j;

    {étape3: la somme finale}
    s := p + (prod / sum);
    calcul_expression := s; {retour de fonction}
end;

Begin
    writeln ('Entrez les valeurs de n, m et p ');
    readln(n, m, p);
    if(m >= 1 and n >= 1)
    begin
        s := calcul_expression; {appel de fonction}
        writeln ('Le résultat est ', s);
    end
    else writeln ('n ou m sont négatives ou nulles !');
End.
```

Exercice 04 Ecrire une procédure dans un programme Pascal qui demande à l'utilisateur le périmètre P de la base circulaire d'un cylindre et sa hauteur H

et qui permet de calculer:
- Le rayon R de la base circulaire.
- L'aire A sachant que $A = 2\pi * R * H$.
- Le volume V du cylindre, sachant que $V = \pi * R * R * H$.

Solution Ex 04

{procédure sans paramètres de calcul RAV d'un cylindre qui emploie les variables globales}

```
program calculs_cylindre;
const pi=3.14; var P, R, A, V : real;

procedure calcul_RAV;
begin
    {calcul le rayon}
    R := P/(2*pi);
    {calcul l'aire}
    A := 2*pi*R*H;
    {calcul le volume}
    V := pi*R*R*H;
end;

Begin
    writeln ('Entrez le périmètre de la base du cylindre:');
    readln(P);
    writeln ('Entrez l'hauteur du cylindre:');
    readln(H);
    if(P >= 0 and H >= 0)
    begin
        calcul_RAV; {appel de procédure}
        {affichage les résultats}
        writeln ('le rayon est ', R);
        writeln('l'aire de la base est ', A) ;
        writeln('le volume du cylindre est ', V);
    end
    else writeln ('P ou H sont négatives !');
End.
```

Exercice 05 Ecrire des programmes Pascal dont chacun demande à l'utilisateur d'entrer trois entiers x, y et z et doit employer une fonction parmi les fonctions de calcul suivantes :
a- le minimum,
b- le maximum,
c- la moyenne,
d- la racine carrée $rc = \sqrt{(x^2 + y^2 + z^2)}$,
de ces trois entiers.

Solution Ex 05-a
{calcul le minimum entre trois entiers}

```
program minimum_trois_entiers;

var x, y, z, min: integer;

function minimum: integer;
begin
    if (x <= y) then
        if (x <= z) then
                        minimum := x
        else  minimum := z
    else
        if (y <= z) then
                        minimum := y
        else  minimum := z  {retour de fonction}
end;

Begin
    writeln('Entrez les entiers x, y et z: ');
    readln(x, y, z);

    {calcul le minimum}
    min := minimum;  {appel de fonction}

    writeln('Le minimum est ', min);
end.
```

Solution Ex 05-b
{calcul le maximum entre trois entiers}

```
program maximum_trois_entiers;
var x, y, z, max: integer;

function maximum: integer;
begin
    if (x >= y and x >= z) then maximum := x;
    if (y >= x and y >= z) then maximum := y;
    if (z >= x and z >= y) then maximum := z; {retour de fonction}
end;

function maximum2: integer;
begin
    max:=x;
    if (max < y) then max := y;
    if (max < z) then max := z;
end;
```

```
Begin
    writeln('Entrez les entiers x, y et z: ');
    readln(x, y, z);
    {solution 1}
    max := maximum; {appel de la première fonction}

    {ou solution 2}
    max := maximum2; {appel de la deuxième fonction}
    writeln('Le maximum est ', max);
End.
```

Solution Ex 05-c

{calcul la moyenne de trois entiers}

```
program moyenne_trois_entiers;
var x, y, z: integer;
    moy: real;
function moyenne: real;
begin
    moyenne := (x+y+z)/3;  {retour de fonction}
end;

Begin
     writeln('Entrez les entiers x, y et z: ');
     readln(x, y, z);
     {calcul la moyenne}
     moy := moyenne; {appel de fonction}
     writeln('La moyenne est ', moy);
End.
```

Solution Ex 05-d

{calcul la racine carrée $rc = \sqrt{(x^2 + y^2 + z^2)}$}

```
program racine_carree;
var x, y, z: integer;
    rc: real;
function calcul_racine_carree: real;
begin
    calcul_racine_carree := sqrt(x^2+y^2+z^2); {retour de fonction}
end;

Begin
     writeln('Entrez les entiers x, y et z: ');
     readln(x, y, z);
     {calcul la racine carrée}
     rc := calcul_racine_carree; {appel de fonction}
     writeln('La racine carrée est ', rc);
End.
```

Exercice 06 Ecrire une fonction dans un programme Pascal qui permet de lire un caractère, puis de le classer dans les catégories suivantes:

Espace: si le caractère est blanc.
Lettre: si le caractère est une lette.
Chiffre: si le caractère est un chiffre numérique.
Spécial: si autre caractère.

Solution Ex 06
{classer un caractère dans (Espace, Lettre, Chiffre, Caractère spécial)}

```
program classer_caractere;
type cat_c = (Espace, Lettre, Chiffre, Caractère spécial);
var c : char;

function classer: cat_c;
var s: cat_c; {variable locale}
begin
    case c of
        'a'..'z' : s := 'Lettre';
        'A'..'Z' : s := 'Lettre';
        '0'..'9' : s := 'Chiffre';
        ' ' : s := 'Espace';
        else s := 'Caractère spécial';
    end; {end case c}
    classer := s; {retour de fonction}
end;

Begin
    write ('Entrez un caractere :');
    readln(c);
    {classement du caractère c}
    writeln('La catégorie du ', c, ' est ', classer); {appel de fonction et affichage}
End.
```

3.3.2 Les fonctions

Exercice 07 Pour un entier positif non nul ($n >= 1$), introduit par l'utilisateur. Ecrire deux programmes Pascal qui appellent à chaque fois une fonction pour calculer les deux sommes suivantes:
a- $S = 1 + 2 + ... + n$.
b- $S = \sum_{i=1}^{n} \sqrt{2i+1}$.

Solution Ex 07-a

{programme avec une fonction qui calcule la somme 1+2+....+ n.}

```
 program somme_des_nombres;
 var i, n, s: integer;

{Déclaration de la fonction somme_f_nombres}
function somme_f_nombres: integer;
var res: integer; begin
    res := 0; {initialisation de la somme}
    for i:=1 to n do res := res+i;
    somme_1_n := res;
end;

Begin
      writeln('Entrez la valeur de n');
      readln(n);
      {somme s=1+2+3+....+n}
      s := somme_for_1_n; {appel de la fonction}

      writeln('s = ', s);
End.
```

Solution Ex 07-b

{programme avec une fonction qui calcule la somme $S = \sum_{i=1}^{n} \sqrt{2i+1}$.}

```
program somme_des_racines;
 var i, n, s: integer;

{Déclaration de la fonction somme_f_racines}

function somme_f_racines: integer;
var res: integer;
begin
    res := 0; {initialisation de la somme}
    for i:=1 to n do res := res+sqrt(2*i+1);
    somme_1_n := res;
end;

Begin
      writeln('Entrez la valeur de n');
      readln(n);
      {somme des racines carrées}
      s := somme_f_racines; {appel de la fonction}

      writeln('s = ', s);
End.
```

3.4 Les procédures et les fonctions avec paramètres

Dans cette section, nous présentons les procédures et les fonctions avec paramètres. A noter aussi que pour certains cas, les résultats sont retournés à travers ces paramètres. Les paramètres employés pour le résultat sont précédés par le mot *var*. Dans la sous-section suivante, nous présentons des exercices qui emploient les procédures avec paramètres.

3.4.1 Les procédures

Exercice 08 Pour un entier positif $m >= 0$ introduit par l'utilisateur, écrire deux programmes Pascal qui permettent de
a- calculer les deux sommes $s1 = \sum_{i=0}^{m}(2i+1)^2$ et $s2 = \sum_{i=0}^{m}(2i+1)^3$. Le résultat de la somme devra être retourné par un paramètre de la procédure.
b- afficher les nombres pairs de 0 à m (en ordre croissant), puis les nombres impairs de m à 0 (en ordre décroissant) en employant la même procédure.

Solution Ex 08-a
{calculer les deux sommes $s1 = \sum_{i=0}^{m}(2i+1)^2$ et $s2 = \sum_{i=0}^{m}(2i+1)^3$.}

```
program somme_deux_expressions;
var i,m, s_carres, s_cubes: integer;

{Déclaration de la procédure somme_puis}
procedure somme_puis(p:integer; var res: integer);
begin
    s := 0; {initialisation de la somme s}
    {solution 1: s=1*1+3*3+....+(2m+1)*(2m+1) avec FOR}
    for i:=0 to m do
        if p=2 then s := s+(2*i+1)*(2*i+1)
        else s := s+(2*i+1)*(2*i+1)*(2*i+1);
    res := s;
end;

Begin
     writeln('Entrez la valeur de m >= 0');
     readln(m);
     somme_puis(2, s_carres); {somme des carrés}
     somme_puis(3, s_cubes); {somme des cubes}

     writeln('somme des carrés est ', s_carres);
     writeln('somme des cubes est ', s_cubes);
End.
```

Solution Ex 08-b
{afficher les nombres pairs de 0 à m et impairs de m à 0.}

```
program affichage_nombres_ordre;
var i, m, k: integer;

{Déclaration de la procédure afficher_ordre}
procedure afficher_ordre(ord: char);
begin
    if ord='c' then begin
     k := 0;
     {les nombres pairs en ordre croissant}
     while k <= m do
     begin
        writeln(' k = ', k);
        k := k + 2;
     end;
    end
    else  begin
     k := m;
     if m mod 2 = 0 k := k - 1;

     {les nombres impairs en ordre décroissant}
     while k >= 1 do
     begin
        writeln(' k = ', k);
        k := k - 2;
     end;
    end;
end;

Begin
     writeln('Entrez la valeur de m >= 0');
     readln(m);
     {afficher les nombres pairs de 0 à m en ordre croissant}
     afficher_ordre('c');
     {afficher les nombres impairs de m à 0 en ordre décroissant}
     afficher_ordre('d');

End.
```

Exercice 09 Ecrire un programme Pascal qui appelle deux procédures chargées de calculer les expressions suivantes:
a- $s = \sum_{i=1}^{n} \prod_{j=0}^{i} (j+1)$.
b- $s = \prod_{i=1}^{n} \sum_{j=0}^{i} (j+1)$.
Pour un entier positif non nul (n >= 1), introduit par l'utilisateur.

Solution Ex 09

{calcul somme des produits et produit des sommes / n >= 1}

```
program somme_produits;

var i, n: integer;
    som, prod: longint;

{Déclaration de la procédure somme des produits pour n >= 1}
procedure somme_p(n: integer; var s: longint);
var i: integer;
    p: longint;

begin
    for i:=1 to n do
    begin
        p := 1;
        for j:=0 to i do
            p := p*(j+1);
        s := s+p;
    end;
end;

{Déclaration de la procédure produit des sommes pour n >= 1}
procedure produit_s(n: integer; var p: longint);
var i: integer;
    s: longint;

begin
    i := 1;
    while i <= n do
    begin
        s := 0;
        j := 1;
        while j <= i do
        begin
            s := s+(j+1);
            j := j+1;
        end;
        p := p*s;
        i := i+1;
    end;
end;

Begin
    writeln('Entrez la valeur de n >= 1');
    readln(n);
```

```
    if(n >= 1) then
    begin
        {appel de la procédure somme des produits}
        som := 0; {initialisation de la somme som}
        somme_p(n, som);

        {ou appel de la procédure somme des produits}
        prod := 1; {initialisation du produit prod}
        produit_s(n, som);
        writeln('somme des produits est ', som);
        writeln('produit des sommes est ', prod);
    end
    else writeln('Erreur: valeur de n est négative ou nulle !');
End.
```

Exercice 10 Une entreprise possède dans son effectif n employés (n un entier positif non nul introduit par l'utilisateur). Ecrire un programme Pascal qui permet d'entrer des salaires de tout les employés pour les 12 mois de l'année courante. Le programme doit permettre de calculer et afficher
a- la masse salariale de 12 mois pour chaque employé (appel d'une procédure).
b- la masse salariale totale.

Solution Ex 10
{calcul et affichage de la masse salariale et le total de n employés}

```
program masse_salariale;
var i, n: integer;
    sal, som, ms: real;

{Déclaration de la procédure de calcul de la masse salariale}
procedure calcul_masse_sal(m: integer; var s: real);
var i: integer;
    sal: real;
begin
    for i:=1 to m do
    begin
        read(sal);
        s := s + sal;
    end;
end;

Begin
    writeln('Entrez le nombre d''employés n');
    readln(n);
    ms := 0; {initialisation du total ms}
    writeln('Entrez la valeur de n >= 1');
    readln(n);
```

```
    if(n >= 1) then
    begin
        {appel de la procédure de calcul de la masse salariale}
        ms := 0; {initialisation de la somme ms}
        for i:=1 to n do
        begin
            som := 0;
            calcul_masse_sal(12, som);
            writeln('totale des salaires pour l''employé ',i, ' est ', som);
            ms := ms+som;
        end;
        writeln('la masse salariale totale est ', ms);
    end
    else writeln('Erreur: valeur de n est négative ou nulle !');
End.
```

3.4.2 Les fonctions

Exercice 11 Ecrire un programme Pascal qui calcule le factoriel des entiers de 0 à 9. Le programme devra appeler une fonction qui calcule le factoriel d'un entier positif.

Solution Ex 11
{programme avec une fonction qui calcule le factoriel.}

```
program dix_factoriels;
var i : integer ;
    f: longint;

{Déclaration de la fonction de factoriel}
function factoriel(n: integer): longint;
var fact: longint; begin
    fact := 1;
    for i:=1 to n do  fact := fact * i;
    factoriel := fact;   { retour de résultat}
end;

Begin { programme principal }
    for i:=0 to 9 do
    begin
        f := factoriel(i);  { appel de la fonction de factoriel}
        writeln('le factoriel de ', i, ' est ', f);
    end;
end.
```

Exercice 12 Reprendre l'exercice précédent (n° 09) de somme des produits et produit des sommes en écrivant un programme Pascal qui doit appeller des fonctions (à la place des deux procédures) chargées de calculer les expressions suivantes:
a- $s = \sum_{i=1}^{n} \prod_{j=0}^{i} (j+1)$.
b- $s = \prod_{i=1}^{n} \sum_{j=0}^{i} (j+1)$.
Pour un entier positif non nul (n >= 1), introduit par l'utilisateur.

Solution Ex 12

{calcul somme des produits et produit des sommes / n >= 1.}

```
program somme_des_produits;
var i, n: integer;
    som, prod: longint;

{Déclaration de la fonction somme des produits pour n >= 1}
function somme_p(n: integer): longint;
var i: integer;
    s, p: longint;
begin
    s := 0;
    for i:=1 to n do
    begin
        p := 1;
        for j:=0 to i do p := p*(j+1);
        s := s+p;
    end;
    somme_p := s;
end;

{Déclaration de la fontion produit des sommes pour n >= 1}
function produit_s(n: integer): longint;
var i: integer;
    p, s: longint;
begin
    p := 1; i := 1;
    while i <= n do
    begin
        s := 0; j := 1;
        while j <= i do
        begin
            s := s+(j+1);
            j := j+1;
        end;
        p := p*s;
        i := i+1;
    end;
    produit_s := p;
end;
```

```
Begin
    writeln('Entrez la valeur de n >= 1');
    readln(n);

    if(n >= 1) then
    begin
        {appel de la fonction somme des produits}
        som := somme_p(n);

        {ou appel de la fonction somme des produits}
        prod := produit_s(nm);

        writeln('somme des produits est ', som);
        writeln('produit des sommes est ', prod);
    end
    else writeln('Erreur: valeur de n est négative ou nulle !');
End.
```

Exercice 13 Reprendre l'exercice précédent (n° 10) de n employés en écrivant un programme Pascal qui doit permettre de calculer et afficher
a- la masse salariale de 12 mois pour chaque employé (appel d'une fonction à la place de la procédure).
b- la masse salariale totale.

Solution Ex 13
{calcul et affichage de la masse salariale et le total de n employés.}

```
program masse_salariale_employes;

var i, n: integer;
    sal, som, ms: real;

{Déclaration de la fonction de calcul de la masse salariale}
function calcul_masse_sal(m: integer): real;
var i: integer;
    s, sal: real;
begin
    s := 0;
    for i:=1 to m do
    begin
        read(sal);
        s := s + sal;
    end;
    calcul_masse_sal := s;
end;
```

```
Begin
    writeln('Entrez le nombre d''employés n');
    readln(n);
    ms := 0; {initialisation du total ms}
    writeln('Entrez la valeur de n >= 1');
    readln(n);
    if(n >= 1) then
    begin
        {appel de la fonction de calcul la masse salariale}
        ms := 0; {initialisation de la somme ms}
        for i:=1 to n do
        begin
            som := calcul_masse_sal(12);
            writeln('totale des salaires pour l''employé ',i, ' est ', som);
            ms := ms+som;
        end;
        writeln('la masse salariale totale est ', ms);
    end
    else writeln('Erreur: valeur de n est négative ou nulle !');
End.
```

Serie TP No3: Les enregistrements et les fichiers

Dans cette série d'exercices, deux nouveaux concepts sont introduits qui sont la structure de l'enregistrement et la manipulation des fichiers (de texte, d'éléments). Pour ce faire, nous proposons des problèmes sous forme d'exercices avec solutions qui sont liés à ces deux concepts.

3.5 Les enregistrements

3.5.1 Type enregistrement

Exercice 01 Déclarer les types qui permettent de stocker et traiter des objets de type enregistrement suivants:

- Un pays caractérisé par
  son nom, son continent d'appartenance, langue, religion dominante, sa population et son superficie.
- Une ville caractérisée par
  son nom, son pays d'appartenance, sa population et son superficie.
- Une date (jour, mois, année).
- Un temps (heure, minute, seconde).
- Un coureur de marathon caractérisé
  par son nom, sa nationalité, pays d'appartenance, sa date de naissance et son meilleur temps.
- Un ensemble de N coureurs de marathon.
- Un marathon caractérisé par
  son nom, les coureurs participants, sa ville d'organisation, sa distance, sa date d'organisation, son temps de démarage.

Solution Ex 01

{sept types à déclarer}

```
Const N = 54;
Type Pays = Record
            nom, continent: string[20];
            langue, religion: string[20];
            population, superficie: integer;
            End;

    Ville = Record
            nom: chaîne[20];
            pays_appartenance: Pays;
            population, superficie: integer;
            End;

    Date = Record
           jour, mois, année: integer;
           End;
    Temps = Record
            heure, minute, seconde: integer;
            End;

    Coureur = Record
              nom, nationalité: string[20];
              pays_appartenance: Pays;
              date_naissance: Date;
              meilleur_temps: Temps;
              End;
    Ens_coureurs = array [1..N] of Coureur;

    Marathon = Record
               nom: string[50];
               coureurs_participants: Ens_coureurs;
               ville_organisation: Ville;
               distance: real;
               date_organisation: Date;
               temps_démarage: Temps;
               End;
```

Exercice 02 Un marathon est caractérisé par un temps de démarage et un temps de fin. Un type Temps qui est composé des champs heure, minute et seconde est proposé pour stocker les données temporelles.
On vous demande d'écrire un programme qui permet d'entrer les temps de démarage et de fin du marathon (de type Temps), puis de calculer la durée de ce marathon (qui est aussi de type Temps).

Solution Ex 02
{programme de marathon}

```
program duree_marathon;
Type
    Temps = Record
        heure, minute, seconde: integer;
    End;

Var d_marathon, f_marathon, duree: Temps;

Begin
    writeln('Entrez le temps de démarage du marathon ');
    readln(d_marathon.heure, d_marathon.minute, d_marathon.seconde);
    writeln('Entrez le temps de fin du marathon ');
    readln(f_marathon.heure, f_marathon.minute, f_marathon.seconde);
    {calcul la durée }
    duree.minute := 0;
    if f_marathon.seconde >= d_marathon.seconde then
        duree.seconde := f_marathon.seconde - d_marathon.seconde
    else
    begin
        duree.seconde := 60 + (f_marathon.seconde - d_marathon.seconde);
        duree.minute := duree.minute -1;
    end;
    duree.heure := 0;
    if f_marathon.minute >= d_marathon.minute then
        duree.minute := f_marathon.minute - d_marathon.minute
    else
    begin
        duree.minute := 60 + (f_marathon.minute - d_marathon.minute);
        duree.heure := duree.heure -1;
    end;
    duree.heure := f_marathon.heure - d_marathon.heure;
    if duree.heure < 0 then
        writeln('les deux temps de démarage et de fin du marathon sont invalides !')
    else
    begin
        writeln('duree.heure = ', duree.heure);
        writeln('duree.minute = ', duree.minute);
        writeln('duree.seconde = ', duree.seconde);
    end;
End.
```

3.5.2 Tableau des enregistrements

Il s'agit ici de deux cas d'implémentation. On implémente un tableau des enregistrements dans le premier cas et un enregistrement des tableaux dans le deuxième cas.

Exercice 03 Pour stocker les notes du module Informatique2 de 1ère année tronc commun ST, un tableau est utilisé pour contenir tous les étudiants inscrits à ce module pendant l'année courante. L'étudiant est caractérisé par les informations suivantes: nom, prénom, numéro d'inscription, et trois notes: TD (20%), TP (20%) et examen (60%). On vous demande d'écrire un programme Pascal qui permet de saisir les informations de tous les étudiants inscrits au module Informatique2, puis de calculer et d'afficher la note moyenne du module.

Solution Ex 03

{tableau des enregistrements}

```
Program gestion_scolarite;
Const nbEtudiants = 150;
      nbNotes = 3;
Type note_t = array [1..nbNotes] of real;
     ponderations_t = array [1..nbNotes] of real;
     etudiant_t = Record
     nom, prenom : string[20];
     numero : integer;
     notes : note_t;
     End;
     classe_t = array [1..nbEtudiants] of etudiant_t;
Var c : classe_t; i, j : integer;
    p : ponderations_t;
    moy : real;
Begin
    for j := 1 to nbNotes do
        readln(p[j]);

    for i := 1 to nb_etudiants do
    begin
        writeln ('Etudiant n.', i);
        readln (c[i].nom);
        readln (c[i].prenom);
        readln (c[i].numero);
        moy := 0;
        writeln (' notes :');
        for j := 1 to nbNotes do
        begin
            readln(c[i].notes[j]);
            moy := moy + p[j]*c[i].notes[j];
        end;
        writeln;
        writeln ('moyenne est ', moy);
    end;
End.
```

Exercice 04 Pour gérer la scolarité de 1ère année tronc commun ST, deux structures de type enregistrement sont employées pour stocker les informations d'un groupe d'étudiants:
- structure pour le groupe: numéro du groupe, tableau des ètudiants inscrits dans ce groupe.
- structure pour l'étudiant: numéro d'inscription, nom et prénom et tableau des notes pour ces modules.
On vous demande d'écrire un programme Pascal qui permet de saisir les informations d'un groupe d'étudiants, puis de calculer et d'afficher la moyenne de chaque étudiant.

Solution Ex 04

{Enregistrement des tableaux }

```
program gestion_scolarite_bis;
const nbEtudiants = 150;
      nbModules = 10;
type etudiant_t = Record
        numero_inscription : integer;
        nom_prenom : array [1..nbEtudiants] of string[50];
        notes : array [1..nbModules] of real;
     End;
     groupe_t = Record
        numero_groupe : integer;
        etudiants : array [1..nbEtudiants] of etudiant_t;
     End;
var g : groupe_t; s, i, j : integer;
Begin
    writeln ('Entrez le numéro du groupe :');
    readln (g.numero_groupe);
    writeln ('Entrez les noms et numéros d''inscription des étudiants de ce groupe:');
    for i := 1 to nb_etudiants do
    begin
        writeln ('Entrer les attributs de l''étudiant ', i);
        readln (g.etudiants[i].numeros_inscription);
        {saisir les notes de l'étudiant i}
        readln (g.etudiants[i].nom_prenom);
        s := 0;
        writeln ('Entrez les notes :');
        for j := 1 to nbModules do
        begin
            readln(g.etudiants[i].notes[j]);
            s := s + g.etudiants[i].notes[j];
        end;
        writeln ('la moyenne de l''étudiant ', i, ' est ', s/nbModules);
    end;
End.
```

3.5.3 Fichiers

On implémente ici les deux types de fichiers de texte et d'éléments.

Exercice 05 On dispose d'un fichier texte, intitulé 'source.txt'. Nous désirons, à partir de ce fichier de texte de lire son contenu et convertir tous les caractères en minuscule, puis le programme sauvegarde le résultat de conversion dans un autre fichier texte nommé 'resultat.txt'. On vous demande de

a- Ecrire un premier programme Pascal qui permet d'effectuer les tâches suivantes :
- accéder aux deux fichiers texte f1 et f2.
- lire tous les caractères, y compris les retours chariots.
- implémenter une fonction **carMinuscule**(c : char) qui convertit les caractères lus en minuscule.
- sauvegarder le résultat de conversion dans le deuxième fichier texte f2.
- lire et afficher tous les caractères du fichier f2 en utilisant le type **string**.

b- Refaire le programme Pascal précédent avec les mêmes tâches en considérant les erreurs d'ouverture des fichiers.

Solution Ex 05-a
{convertir un fichier texte en minuscule et sauvegarder le résultat de conversion.}

```
program conversion_fichier_texte;
const nomf1='c:\st\source.txt';
      nomf2 ='c:\st\resultat.txt';
type ligne_t = string[127];
var f1, f2 : text;
    car: char; mot: ligne_t;

function carMinuscule(c: char) : char;
begin
    if (c >= 'A') and (c <= 'Z') then carMinuscule := chr(ord(c) - ord('A') + ord('a'))
    else carMinuscule := c;
end;

Begin
    {ouverture les fichiers en lecture et en écriture}
    assign (f1, nomf1);
    assign (f2, nomf2);
    reset (f1);
    rewrite (f2);

    while not eof(f1) do    {lecture de tous les caractères, avec recopiage en minuscule}
    begin
        read (f1, car);
        write (f2, carMinuscule(car));
    end;
```

```
    while not eof(f2) do    {lecture et affichage du fichier texte f2}
    begin
        readln(f2, mot);
        writeln (mot);
    end;

    {fermeture des fichiers}
    close(f1);
    close(f2);
End.
```

Solution Ex 05-b

{convertir un fichier texte en minuscule et sauvegarder le résultat de conversion avec traitement d'erreurs d'ouverture de fichiers.}

```
program conversion_fichier_texte_bis;
{...}
     err : integer;

Begin
    {ouverture les fichiers en lecture et en écriture}
    assign (f1, nomf1);
    assign (f2, nomf2);

    {$I-} reset (f1); {$I+}
    if IoResult <> 0 then
    begin
        err := 1;
        writeln ('Erreur d''ouverture de fichier texte f1 !');
    end
    else
    begin
        {$I-} rewrite (f2); {$I+}
        if IoResult <> 0 then
        begin
            err := 2;
            writeln ('Erreur d''ouverture de fichier texte f2 !');
        end
        else
        begin
            {lecture de tous les caractères, avec recopiage en minuscule}
            while not eof(f1) do
            begin
                read (f1, car);
                write (f2, carMinuscule(car));
            end;
```

```
            {lecture et affichage du fichier texte f2}
            while not eof(f2) do
            begin
                readln(f2, mot);
                writeln (mot);
            end;
        end;
    end;

    {fermeture des fichiers}
    close(f1);
    close(f2);
End.
```

Exercice 06 Un médecin veut informatiser la gestion de son cabinet. On propose de l'aider à travers un programme Pascal qui emploie deux structures pour stocker les informations des patients:
- structure pour le patient: numéro de patient, nom et prénom, date de naissance et date du prochain RDV,
- structure pour la date: (jour, mois, année).
On vous demande d'écrire un programme Pascal qui permet de d'effectuer les tâches suivantes :
- créer un fichier de type élément.
- saisir continuellement les patients tant qu'on pas introduit un zéro pour le numéro du patient.
- afficher la liste des patients.

Solution Ex 06

{gestion de cabinet pour stocker les informations d'enregistrements dans un fichier de type élément.}

```
program gestion_cabinet;

const nomf ='c:\st\patients.txt';
type date_t = Record
                jour, mois, annee : integer;
              End;
     patient_t = Record
                    numero_patient : integer;
                    nom : string[25];
                    prenom : string[25];
                    date_naissance, date_RDV : date_t;
                End;
var f : file of patient_t; p : patient_t;

Begin
    {ouverture de fichier en lecture et en écriture}
    assign (f, nomf);
    rewrite (f);
```

```
    {introduire les patients dans le fichier f}
    repeat
        write('Numéro du patient : ');
        readln(p.numero_patient);
        if(p.numero_patient <> 0) then
        begin
            write('Nom et prénom : ');
            readln(p.nom, p.prenom);
            write('Date de naissance : ');
            readln(p.date_naissance.jour, p.date_naissance.mois, p.date_naissance.annee);
            write('Date de prochain RDV : ');
            readln(p.date_RDV.jour, p.date_RDV.mois, p.date_RDV.annee);

            {ecrire les données dans le fichier f}
            write(f, p);
        end;
    until p.numero_patient = 0;

    {afficher la liste des patients depuis le fichier f}
    while not eof(f) do
    begin
        read(f, p);
        writeln (p.numero_patient, ' ', p.nom, ' ', p.prenom, ' né le (',
        p.date_naissance.jour, p.date_naissance.mois, p.date_naissance.annee,') ',
        ' avec RDV (', p.date_RDV.jour, p.date_RDV.mois, p.date_RDV.annee, ') ');
    end;
    {fermeture de fichier}
    close (f);
End.
```

Conclusion

Ce polycopié du module 'Informatique2' s'adresse aux étudiants de première année de sciences et technologie (ST). Il est présentée au profit des étudiants ayant des difficultés à étudier l'algorithmique et la programmation Pascal. Comme il est mentionné dans l'introduction, ce polycopié est scindé en deux parties: la première partie est consacrée pour présenter les concepts fondamentaux comme les tableaux, les fonctions et les procédures et aussi les fichiers avec des notions théoriques de base pour l'algorithmique et la programmation Pascal.

Une deuxième partie est préparée et organisée sous forme de séries contenant des travaux pratiques (exercices avec solutions). En fin, j'espère que le contenu de ce polycopié a bien aidé et assisté l'étudiant dans sa formation académique et aussi lui permettant de se familiariser avec les langages de programmation. Pour cela trois séries d'exercices avec des solutions sont proposés avec des explications riches et pertinentes.

Bibliographie

[1] R. Amour, R. Fermous, M. Ait Ouarabi, and M. Benzekka. Algorithmique : Cours et exercices en programmation pascal. Cours, Université de sciences et technologie Houari Boumediene, Alger, Algérie, 2018.

[2] Jean-Michel DOUDOUX. Le langage pascal objet, consulté le 10 jan. 2023. "https://www.jmdoudoux.fr/delphi/langage.html".

[3] PMTIC. Environnement numérique. "pmtic_env_num_machine_ordi.pdf".

[4] L'équipe Superprof. Historique des programmes informatiques, consulté le 10 fév. 2023. "https://www.superprof.fr/blog/historique-programmes-informatiques/".

[5] T.Dumartin. Architecture des ordinateurs. Note de cours, informatique industrielle, 2005.

[6] O. Thiare. Algorithmes et programmation. Cours, UFR de sciences appliquées et de technologie, Université Gaston Berger, Saint-Louis, Sénégal, 2008.

[7] E. Thiel. Algorithmes et programmation en pascal. Cours deug 1 mass ma, Faculté des Sciences de Luminy, Marseille, 2004.

[8] M. Volle. Histoire du micro-ordinateur, consulté le 10 fév. 2023. "http://www.volle.com/ulb/021122/textes/histoiremicro.htm".