Cours et exercices corrigés en Pascal

Jean Marc Salotti

Professeur des Universités
UFR SM / Université Bordeaux 2

1998

SOMMAIRE

Introduction

L'informatique a pris une place importante dans la société d'aujourd'hui : avec des ordinateurs de plus en plus puissants et des réseaux qui permettent un transfert planétaire des informations, il n'est plus possible d'ignorer le monde informatique. D'un autre côté, l'univers informatique s'étendant au travers d'une multitude de technologies aussi différentes que complexes, même le plus érudit des informaticiens est inévitablement ignorant dans une des nombreuses branches du domaine informatique.

En DEUG A, les premiers cours d'informatique commencent traditionnellement par des notions d'algorithmique et l'apprentissage d'un langage de programmation. Cette première approche est discutable car elle implique un grand nombre d'impasses et les étudiants ne perçoivent paradoxalement qu'un monde relativement étriqué. La programmation est cependant une composante essentielle de l'informatique, car elle est le moyen de son développement, que ce soit au niveau de la microprogrammation, des systèmes d'exploitation ou des grands logiciels. La recherche du programme réalisant le traitement adéquat est également un des aspects les plus intéressants de l'informatique. Cette recherche nécessite beaucoup de réflexion et constitue en plus un travail de création souvent original.

Ce manuel résume le cours informatique de DEUG A première année. Il est composé de 12 sections correspondant approximativement à 12 semaines de cours, travaux dirigés et travaux pratiques. Il comprend un grand nombre d'exemples et d'exercices corrigés qui pourront aider l'étudiant dans son travail.

1 Introduction à l'informatique

L'informatique est la science du traitement automatique des informations. Cette définition est un peu vague car elle fait appel aux deux mots abstraits "information" et "traitement".

En informatique, une **information** peut être de nature très diverse. On définit en général deux grandes catégories d'information : la première est l'information numérique, par exemple l'âge d'une personne ou le prix d'une voiture, la seconde est l'information symbolique comme par exemple la description d'une personne ou la couleur d'un mur. Quelle que soit l'information à traiter, **on se ramène toujours à une information numérique** en attribuant une valeur à chaque information. Par exemple, si on a l'ensemble de couleurs {rouge, vert, bleu}, on peut proposer le codage suivant : 1<--> rouge; 2 <--> vert; 3 <--> bleu.

En codant toutes les informations par des valeurs numériques, le **traitement** des informations se ramène toujours à un traitement sur des nombres. Ce traitement numérique peut être une opération mathématique classique (addition, multiplication...), une comparaison entre 2 nombres, une mémorisation ou un transfert de données. Aussi complexe que soit le traitement, celui-ci n'est rien d'autre qu'une combinaison de ces opérations élémentaires !

1.1 Les grands domaines de l'informatique

Le développement de l'informatique a conduit à une division en plusieurs domaines. Ces domaines sont à présent si vastes qu'il est pratiquement impossible pour une seule personne de devenir un spécialiste de toute l'informatique. Sans être exhaustif, on peut citer les domaines suivants :

• L'architecture des ordinateurs
Il s'agit de toutes les connaissances relatives à la conception des circuits, des cartes, de tous les composants électroniques, en un mot du "hardware" (ce qui est dur, que l'on peut donc sentir au toucher). On parle par exemple de l'architecture du 68000, du Pentium ou du PowerPC.

• Les systèmes d'exploitation
L'écran, le clavier, la souris, le lecteur de disquette, le disque dur (entre autres) sont appelés les ressources de l'ordinateur. Or, pour exploiter ces ressources, il faut faire appel à des protocoles compliqués. Le système d'exploitation n'est rien d'autre qu'un programme qui gère l'interface entre l'ordinateur et l'utilisateur. A l'aide de commandes simples, on peut ainsi savoir la place restant en mémoire ou ce qui est en cours d'impression. On peut citer les systèmes d'exploitation MSDOS, Windows95, OS/2, Unix, Mac OS….

• La Bureautique
La Bureautique est l'ensemble des outils informatiques utilisés pour automatiser des tâches administratives ou de secrétariat. Parmi ces outils, le traitement de texte et le tableur sont les plus connus. Le traitement de texte permet de composer et mettre en page un texte de façon très conviviale. Le tableur permet d'effectuer des traitements sur des informations rangées dans un tableau, par exemple la moyenne ou la somme d'une série de nombres.

• Les bases de données
Une base de données est un ensemble organisé d'informations. L'exemple le plus connu est celui de l'annuaire téléphonique accessible sur Minitel. Cette base de données est composée

d'une longue liste de fiches où sont mentionnés l'identité de la personne, son adresse et son numéro de téléphone. De plus en plus de commerçants utilisent une base de données pour répertorier les produits qu'ils vendent, en mémorisant la référence, les propriétés du produit (couleur, taille…), le prix d'achat, le prix de vente, le nom du fournisseur, etc.. En précisant la référence, ils accèdent directement aux caractéristiques du produit qu'ils peuvent alors faire apparaître sur la facture automatiquement.

• Les réseaux

Un réseau informatique est un ensemble d'ordinateurs reliés entre eux. Les câbles sont soient dédiés au réseau informatique, soient les mêmes que ceux utilisés pour le téléphone, c'est le cas du Minitel, soient inexistants si on utilise par exemple une liaison laser ou un satellite. La liaison peut être locale (un réseau au sein d'une même entreprise par exemple) ou plus globale comme Internet qui est un réseau planétaire. Grâce à un protocole de communication entre ordinateurs standardisé, des ordinateurs de différents types peuvent échanger des données. Internet et les réseaux en général offrent de nombreux services : chaque ordinateur ayant une adresse sur le réseau, on peut envoyer un message à un ami ou recevoir le sien qui est alors stocké, en attente d'être lu, dans ce qu'on appelle une boîte aux lettres électronique. On peut bien entendu accéder à un très grand nombre d'informations (exemple : le Minitel) et les récupérer sur notre ordinateur. Si on dispose des droits d'entrée requis, on peut également se connecter sur un ordinateur éloigné et exécuter un programme à distance. Ce dernier service est intéressant si l'ordinateur distant dispose de ressources qui n'existent pas localement.

• Les langages informatiques

Toutes les applications informatiques (le traitement de texte, les jeux, …) sont conçues dans un langage de programmation, avec un vocabulaire et une grammaire spécifique. Certains langages sont dit évolués car ils permettent d'effectuer des traitements, non seulement sur des nombres, mais aussi sur des caractères, des tableaux ou d'autres structures plus complexes. Une fois que le programme est écrit, il ne peut fonctionner directement car, comme nous l'avons déjà dit, tous les traitements doivent être numériques. Pour qu'il fonctionne, on utilise un outil qui s'appelle "compilateur" et qui traduit en "langage machine" (c'est à dire en opérations élémentaires) toutes les données et tous les traitements. Il existe un grand nombre de langages informatiques, par exemple le Pascal, le C, le C++, le Fortran, le LISP, etc..

• L'automatique[1]

L'automatisation des tâches nécessite l'automatisation du traitement de l'information, mais pas obligatoirement un ordinateur. Parmi les différents domaines de l'automatique, la robotique ne fait pas partie de l'informatique, mais elle y est cependant très liée. En effet, pour piloter un robot, on travaille souvent avec un ordinateur dans lequel on a placé un langage de commandes dédié à la manipulation du robot. L'intérêt de l'ordinateur et de ce langage est d'adapter les mouvements du robot à une tâche spécifique et de pouvoir en changer si on le désire, sans être obligé de reconfigurer ou même de changer le matériel. A l'opposé, une machine à laver le linge a souvent plusieurs options de programmation, mais il est exclu de laisser à l'utilisateur une liberté totale sur les opérations à effectuer. Dans ce cas, une simple puce électronique, ne sachant exécuter qu'un nombre limité d'opérations convient bien mieux qu'un ordinateur.

[1] Nous parlons ici de l'automatique afin de mieux cerner les frontières de l'informatique.

• Les jeux
Il existe une multitude de jeux faisant intervenir la réflexion ou l'adresse. La programmation d'un jeu de réflexion, comme les échecs ou les dames, nécessite la mise à plat de l'expertise d'un joueur, ce qui est intéressant car les joueurs ont souvent bien du mal à expliquer le choix de leur coup. En ce qui concerne les jeux d'arcades où un personnage central doit combattre d'autres créatures et se déplacer dans un environnement peu hospitalier, la programmation passe par une modélisation de chaque lieu, de chaque stature du personnage et des propriétés contextuelles de chaque objet. Dans un environnement en trois dimensions, les calculs de repositionnement de la scène et des objets en mouvements doivent prendre en compte tellement d'informations qu'il est nécessaire d'avoir un ordinateur très puissant … ou de simplifier la scène. Les jeux de demain se feront dans des univers virtuels : muni d'un casque disposant de propriétés similaires à une manette de jeu et de gants tactiles enregistrant les déplacements de la main et la force de préhension des doigts, l'être humain sera plongé dans un monde imaginaire très réaliste.

• L'intelligence artificielle
Il n'existe pas de définition universellement reconnue de l'intelligence artificielle car la notion même d'intelligence est mal définie. Si on prend comme référence les capacités humaines, aucun ordinateur actuel ne peut être raisonnablement qualifié d'intelligent. Les difficultés rencontrées sont liées à la complexité du monde dans lequel nous vivons, qui est bien difficile à décrire et à la complexité de l'intelligence humaine, bien difficile à appréhender. Les chercheurs tentent de reproduire toutes les capacités intellectuelles humaines, tel l'apprentissage ou la vision par ordinateur (en utilisant une caméra), mais on doute d'arriver un jour à faire aussi bien que notre cerveau.

1.2 Le fonctionnement des ordinateurs

Le fonctionnement des ordinateurs est devenu aujourd'hui très complexe. Nous allons le présenter de façon très simplifié en présentant ses principaux composants.

On peut distinguer six éléments essentiels :
- Le processeur
- La mémoire de travail
- L'écran, le clavier et la souris
- Les disquettes et le disque dur
- La carte mère, les cartes spécialisées et les bus
- Les autres périphériques

Le processeur
Le processeur est une sorte de petite boîte magique capable d'effectuer une opération sur des nombres. Il s'agit en général d'une opération unaire ou binaire, c'est à dire 1 ou 2 nombres en entrée de la boîte et 1 nombre résultat en sortie. Parmi ces opérations, il y a l'addition et la soustraction, la multiplication par 2 et les tests (par exemple on décide de mettre la sortie à 1 si le premier nombre est plus grand que le second et à 0 sinon). Le processeur ne réalise qu'une seule opération à la fois, on dit que les opérations sont "séquentielles". Tout le problème est en fait de définir quelle est la suite d'opérations que l'ordinateur doit exécuter. Un programme exécutable est la liste ordonnée des codes d'opération (et éventuellement les paramètres de ces opérations) que doit exécuter l'ordinateur pour réaliser une tâche spécifique. Le temps d'exécution d'une opération est de l'ordre de quelques nanosecondes. Lorsqu'on

parle d'un ordinateur dont la fréquence d'horloge est de 200 MHz, cela veut dire que le processeur exécute pratiquement une opération toutes les 1/200 millionièmes de seconde, ce qui est extrêmement rapide !

La mémoire de travail

La mémoire de travail ou mémoire vive (souvent assimilée à la RAM, abréviation de Random Access Memory) est utile au fonctionnement du processeur. En effet, c'est dans cette mémoire que sont stockés les programmes en instance d'exécution, sous la forme d'une suite de codes d'opérations élémentaires. Il y a aussi un tas d'informations concernant les calculs en cours, l'état des ressources (position de la souris, caractéristiques des fenêtres apparaissant à l'écran, etc.). C'est une mémoire "vive" et "de travail" car le processeur travaille constamment avec elle, mémorisant, récupérant, effaçant ou remplaçant tour à tour des données. Au démarrage de l'ordinateur, le premier programme à être lancé est **le système d'exploitation** qui utilise cette mémoire pour faire le point sur l'état des ressources de l'ordinateur et préparer l'interface avec l'utilisateur.

L'écran, le clavier et la souris

Que ce soit l'écran de l'ordinateur ou de la télévision, celui-ci est décomposé en un grand nombre de points qu'on appelle "pixels". Ces pixels sont rangés en lignes et colonnes pour former un rectangle de points de taille variable, par exemple 800 en largeur et 600 en hauteur. La couleur de chaque point est mémorisée dans une "carte vidéo" placée dans l'ordinateur. L'écran est un périphérique pour l'ordinateur (c'est à dire un élément qui est rajouté autour de l'ordinateur), au même titre que l'imprimante, le clavier ou la souris. L'écran n'est d'ailleurs pas un composant indispensable, la preuve, on peut l'éteindre et le rallumer sans perturber le fonctionnement de l'ordinateur. En ce qui concerne le clavier et la souris, ils sont en liaison directe avec le processeur : dés qu'une touche est enfoncée ou qu'un déplacement a lieu, un message est envoyé au processeur qui agit immédiatement en conséquence.

Les disquettes et le disque dur

Les disquettes et le disque dur sont des mémoires de stockage. Même lorsque l'ordinateur est éteint, les données sont préservées. Elles servent notamment à stocker des **fichiers**, c'est à dire des ensembles compacts de données auxquels on a donné un nom. Il y a par exemple des fichiers textes qui ont été écrits avec un traitement de texte, des programmes ou des logiciels achetés dans le commerce, mais aussi tout un tas de fichiers systèmes servant à la gestion de l'ordinateur et de ses ressources. Pour des raisons de cohérence, les fichiers sont regroupés suivant leur fonction ou leur provenance dans des ensembles qu'on appelle **répertoires**. Ces répertoires sont parfois également regroupés à l'intérieur d'autres répertoires, si bien que les répertoires et les fichiers forment une arborescence. Pour spécifier un fichier, il faut alors donner le chemin des répertoires qui mènent jusqu'à lui. La gestion des fichiers, déplacement, copie sur disquette, création de répertoire, suppression, etc., est un des rôles essentiels assuré par le **système d'exploitation**. Un disque dur est une mémoire de grande capacité par rapport à la mémoire vive. On pourrait se demander pourquoi le processeur ne se servirait pas plutôt de cette mémoire pour travailler. Le problème, c'est que les temps d'accès et de lecture sur le disque dur sont environ mille fois plus lents qu'avec la mémoire vive, dédiée au processeur. Une disquette a les mêmes propriétés qu'un disque dur. Elle est bien entendu plus facile à transporter, mais en contrepartie, elle a une capacité mémoire beaucoup plus petite.

La carte mère, les cartes spécialisées et les bus
Un ordinateur est un assemblage de plusieurs éléments. La carte mère est une plaque rectangulaire sur laquelle se greffent tous les composants de l'ordinateur. Il y a par exemple des encoches prévues pour des barrettes de mémoire vive et d'autres pour le processeur. Il y a également des fentes pour encastrer d'autres cartes perpendiculairement à la carte mère, par exemples la carte vidéo pour la gestion de l'écran, la carte contrôleur du disque dur et du lecteur de disquettes, la carte son, la carte réseau, etc.. L'intérêt de cette conception modulaire réside dans l'interchangeabilité de chaque élément. Par exemple, il est possible de changer de carte vidéo pour disposer d'un plus grand nombre de couleurs par pixel sans toucher aux autres composants. De même, dans une certaine mesure, on peut remplacer le processeur par un autre plus puissant. Le seul problème réside dans l'échange de données entre le processeur et toutes ces cartes. Il se fait grâce à des circuits imprimés sur la carte mère et qui portent le nom de bus. Par exemple, lorsqu'un changement doit apparaître à l'écran, le processeur fait son calcul et envoie ses directives à la carte vidéo via le bus.

Les autres périphériques
Il existe beaucoup d'autres périphériques, comme le lecteur de cédéroms, l'imprimante, le scanner, le modem, la manette de jeu, etc.. La conception des ordinateurs étant modulaire, il n'y a en fait aucun problème théorique pour connecter n'importe quel appareil électrique à un ordinateur. Il y a en général deux cas de figure pour l'installation d'un périphérique. Si celui-ci peut se brancher sur une prise standard déjà présente à l'arrière de l'ordinateur, il suffit de disposer du logiciel adéquat pour l'utiliser. Dans le cas contraire, il est nécessaire d'ouvrir l'ordinateur et de rajouter la carte contrôleur de ce périphérique en l'insérant dans une fente libre de la carte mère. C'est ainsi qu'on peut contrôler la tension d'un voltmètre, la température d'un four ou piloter le bras d'un robot.

1.3 Le codage des informations

Pour effectuer des calculs, il faut un moyen de représenter les nombres. Il est possible de prendre comme valeur la tension exprimée en volts ou en millivolts, mais le moindre défaut ou la moindre résistance serait la cause de nombreuses erreurs. Pour simplifier le problème, on a décidé de ne considérer que 2 mesures possibles, quand la tension est nulle et quand elle ne l'est pas. A ces 2 mesures on associe le 0 et le 1 et on représente tous les nombres dans le système binaire, qui est décrit ci-dessous.

De même qu'en décimal on groupe par 10 pour accéder au rang supérieur, en binaire on groupe par 2. Cette décomposition est décrite ci-dessous pour les nombres de 1 à 8 :

		Binaire	Décimal
X	1 unité	1	1
(XX)	1 groupe de 2, 0 unité	10	2
(XX) X	1 groupe de 2 et 1 unité	11	3
((XX) (XX))	1 groupe de 4, 0 groupe de 2, 0 unité	100	4
((XX) (XX)) X	1 groupe de 4, 0 groupe de 2, 1 unité	101	5
((XX) (XX)) (XX)	1 groupe de 4, 1 groupe de 2, 0 unité	110	6
((XX) (XX)) (XX) X	1 groupe de 4, 1 groupe de 2, 1 unité	111	7
(((XX) (XX)) ((XX) (XX)))	1 groupe de 8, 0 de 4 et de 2, 0 unité	1000	8

etc.…
Pour exprimer un nombre, on le décompose donc en puissance de 2.

Exemple : $235 = 2 \times 10^2 + 3 \times 10^1 + 5 \times 10^0$

$= 1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

En binaire, 235 s'écrit donc : 11101011

Pour passer d'un nombre décimal en son équivalent binaire, il faut procéder à des divisions successives par 2 pour trouver chacun des chiffres.

Exemple :
235 / 2 = 117 et il reste 1 unité
117 / 2 = 58 et il reste 1 groupe de 2
58 / 2 = 29 et il reste 0 groupe de 4
29 / 2 = 14 et il reste 1 groupe de 8
14 / 2 = 7 et il reste 0 groupe de 16
7 / 2 = 3 et il reste 1 groupe de 32
3 / 2 = 1 et il reste 1 groupe de 64
1 / 2 = 0 et il reste donc pour finir 1 groupe de 128
Pour reformer le nombre binaire, il faut remettre dans l'ordre les chiffres des restes :
11101011 ce qui est bien le nombre trouvé plus haut.

Les entiers sont ainsi codés par l'ordinateur, mais il se pose toutefois le problème des grands nombres : la mémoire de l'ordinateur n'est pas infinie et il n'est donc pas possible de coder tous les nombres. Dans le cas général, la mémoire de l'ordinateur est divisée en octets, c'est à dire en groupe de 8 bits, 1 bit étant un chiffre 0 ou 1. Avec 8 bits, on peut coder les entiers de 0 à 2^8-1, c'est à dire 255, ce qui est peu. Dans les cas les plus courants, les ordinateurs peuvent travailler sur 2 ou 4 octets ce qui permet de coder les entiers jusqu'à 2^{32}, soit un peu plus de 4 milliards.

Remarques :

• Pour additionner 2 nombres binaires, on procède comme en décimal, en n'oubliant pas que 1+1 ne fait pas 2 mais 10, ce qui implique une retenue au rang supérieur.

• Pour coder les nombres négatifs, il existe plusieurs méthodes, le principe de base étant qu'on réserve 1 bit particulier pour le signe (par exemple 0 si c'est positif et 1 sinon).

• On ne peut pas représenter tous les réels, car il faudrait une précision infinie, ce qui est impossible. On se contente donc des nombres décimaux avec une précision limitée. Pour coder ces nombres, on se ramène généralement à l'expression suivante (norme IEEE) :

X étant un réel, $X = 1,\text{mantisse} \times 2^{\text{exposant}}$

Il ne reste alors qu'à coder la mantisse qui correspond aux chiffres placés juste après la virgule et l'exposant de la puissance de 2.

Exemple : $13,5 = 1,6875 \times 2^3$

Or $0,6875 = 0,5 + 0,125 + 0,0625 = 1x2^{-1} + 0x2^{-2} + 1x2^{-3} + 1x2^{-4}$

Le codage binaire de 13,5 sur 32 bits (en simple prÈcision) est donc :

0 0 0 0 0 0 0 1 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Signe Exposant sur 8 bits Mantisse sur 23 bits

• Pour coder les lettres de l'alphabet et tous les caractères en général, il suffit de choisir une convention pour associer un nombre binaire et un caractère. Le codage ASCII est la convention la plus connue. Exemple : 'A' correspond à 65, 'B' à 66, '3' à 51, '!' à 33, etc..

• Attention aux unités : 1 octet = 8 bits, 1Ko = 2^{10} octets = 1024 octets
et 1 Mégaoctet, en abrégé 1Mo = 2^{20} octets, soit 1 048 576 octets.

1.4 Exercices

1. Quel est le rôle d'un système d'exploitation ?

2. Pourquoi le disque dur n'est-il pas utilisé comme mémoire de travail par le processeur ?

3. Si on code la couleur d'un pixel sur 24 bits et que l'écran fait 800 colonnes par 600 lignes, quelle doit être la taille minimale, exprimée en octets, de la mémoire vidéo ?

4. Si on considère qu'il existe 200 caractères accessibles depuis le clavier, combien faut-il de bits au minimum pour pouvoir attribuer un code binaire unique à chacun d'entre eux ? Si on dispose d'un disque dur de 500 Mégaoctets et qu'on suppose qu'une page de livre comprend environ 400 caractères, combien de pages peut-on stocker sur le disque dur ?

5. Quelle est la valeur binaire du nombre décimal 766 ?

6. Quelle est la valeur décimale du nombre binaire 101010101 ?

7. Effectuez l'addition binaire des nombres 101110 et 11100. Vérifiez le résultat en décimal.

8. Exprimez -29,5 en binaire, selon le format proposé au 1.3. Trouvez une méthode systématique pour trouver l'exposant et la mantisse à partir de l'expression décimale. Essayez avec 28,4. Que constatez vous ?

1.5 Solutions

1. Voir la définition du terme au 1.1 et la définition du disque dur au 1.2.

2. Pour une question de rapidité. Il faut noter toutefois que le système d'exploitation peut dans certains cas récupérer un peu de place sur le disque dur dans le but d'augmenter la taille de la mémoire de travail. On dit dans ce cas qu'on dispose d'une mémoire de travail virtuelle.

3. 1 octet = 8 bits, donc la taille minimale est 800x600x24/8 = 1 440 000 octets.

4. La première puissance de 2 au-dessus de 200 est 256 soit 2^8, il faut donc 8 bits, soit 1 octet.
500 Mo / 400 octets = 500 x 1 048 576 / 400 = 1 310 720 pages !

5. 766 en décimal vaut 1 0 1 1 1 1 1 1 1 0 en binaire.

6. 101010101 en binaire vaut 256+64+16+4+1 = 341 en décimal

7. Le résultat de l'addition est : 1001010, soit 46 + 28 = 74 en décimal.

8. Décomposons -29,5 :

-29,5 = 1,84375 x 2^4 -> l'exposant est donc 4, soit 0000 0010 en binaire
0,84375 = 0,5+0,25+0,0625+0,03125 soit donc 11011 suivi de 18 zéros pour la mantisse
Avec 1 devant pour le négatif, le résultat est : 1 00000010 1101100 00000000 00000000

Méthode systématique pour coder un nombre X décimal quelconque :
Pour trouver l'exposant, il faut prendre la partie entière du logarithme base 2 du nombre décimal. Cela revient en fait à trouver la première puissance de 2 inférieure au nombre.
Pour trouver la mantisse, il faut procéder à une série de multiplications par 2 :
soit 0,m le résultat de X / 2^{exposant} - 1,
si 0,m x 2 > 1, alors le 1er chiffre m1 est 1, c'est à dire 1x2^{-1}, sinon m1 = 0
si (0,m x 2 -m1) x 2 > 1, alors le 2ème chiffre m2 est 1, c'est à dire 1x2^{-2}, sinon m2 = 0
si ((0,m x 2 -m1) x2 -m2) x 2 > 1, alors le 3ème chiffre m3 est 1, pour 1x2^{-3}, sinon m3 = 0
Etc.…
Avec 28,4 la méthode des multiplications par 2 n'aboutit pas et on obtient une série infinie de chiffres binaires. Avec 23 chiffres pour la mantisse, l'approximation est très bonne, mais le codage ne permet pas une correspondance exacte.

2 Les langages, introduction au Pascal

2.1 Introduction aux langages

Un langage est composé des éléments suivants :

- Un ensemble de caractères. Pour le Français, il s'agit des lettres de l'alphabet, minuscules et majuscules et des signes de ponctuation.

- Un vocabulaire. Un mot étant un ensemble ordonné de caractères, un vocabulaire défini l'ensemble des mots autorisés. Pour le Français, ce sont les mots du dictionnaire, en incluant les différentes possibilités de conjugaison et les accords féminin ou pluriel.

- Une syntaxe, autrement dit un ensemble de règles grammaticales. En Français, nous avons notamment des règles de conjugaison pour accorder le verbe et le sujet, des règles pour positionner l'article par rapport au nom, des règles pour accorder l'adjectif avec le nom auquel il se rapporte, des règles pour introduire des propositions subordonnées, etc..
Exemples :

Malgré sa blessure, le chien ronge un os de poulet. a une syntaxe correcte.
Malgré sa blessure, le chien rongerons un os de poulet. a une syntaxe incorrecte.
Malgré sa blessure, un os de poulet ronge le chien. a une syntaxe correcte.
Le chien, malgré sa blessure, ronge un os de poulet. a une syntaxe correcte.

- Une sémantique, autrement dit des règles de bon sens. En Français, il y a des phrases qui ont un sens et d'autres non. Ainsi, "Un os de poulet ronge le chien" n'a pas de sens, on dit que la phrase est sémantiquement fausse.

- Il existe également dans les langages ce qu'on appelle une pragmatique, qui exprime le fait qu'il doit normalement y avoir une cohérence dans les idées. Ainsi, si je dis "Il faut travailler pour avoir de bonnes notes et le chien, malgré sa blessure, ronge un os de poulet", j'exprime deux réalités sémantiquement correctes, mais qui marquent une incohérence de l'idée générale.

En informatique, les langages ont également un ensemble de caractères (un alphabet), un vocabulaire et un ensemble de règles syntaxiques qu'il faut respecter pour réaliser un programme. Il est important de bien comprendre ces notions, car la programmation est un exercice de style difficile où les erreurs de langage sont courantes. Si en Français, le partenaire comprend le sens de votre phrase même si celle-ci n'est pas tout à fait correcte grammaticalement, en informatique, la moindre erreur de vocabulaire ou de syntaxe nécessite une relecture et une correction.

Remarquons pour finir qu'un programme écrit dans un langage informatique n'est qu'une suite de mots et n'est donc pas directement compréhensible par le processeur. Comme nous l'avons déjà dit au 1.1, il faut transformer cette suite de mots en une suite d'opérations élémentaires que peut exécuter le processeur. Cette étape s'appelle la compilation. C'est lors de la compilation que les erreurs de syntaxe sont détectées. Heureusement pour nous, le compilateur donne en général suffisamment d'indications pour que la correction de nos fautes

soit simple et rapide. Ensuite, il reste à tester notre programme et à vérifier que celui-ci réalise bien le traitement désiré, autrement dit si sa sÈmantique est correcte.

2.2 Structure générale des programmes Pascal

L'alphabet du Pascal est tout simplement l'ensemble des caractères du clavier. Nous allons voir un certain nombre de mots du vocabulaire, mais il est important de noter, avant de poursuivre, que l'espace et le retour à la ligne sont des séparateurs (ils séparent les mots) et que la syntaxe en autorise autant qu'on le désire. En conséquence, on peut aérer les programmes pour en faciliter la lecture !
D'autre part, tout ce qui est entre accolades est considéré comme du commentaire et est ignoré par le compilateur. Cela va nous servir pour commenter nos programmes et expliquer chaque étape.

Un programme Pascal respecte toujours la structure suivante :

```
program nomprog ;
{ Définition éventuelle des constantes et des types; cette partie est expliquée au chapitre 7 }

{ Début de la déclaration des variables }
var
        nomvar1, nomvar2, … : type1 ;
        nomvar3 : type2 ;
        …                               { Il peut y en avoir autant qu'on veut }
{ Fin de la déclaration des variables }
{ Début du corps du programme }
begin
        instruction1 ;
        instruction2 ;
        …                               { Il peut y en avoir autant qu'on veut }
end.
{  Fin du programme }
```

- "program", "type", "var", "begin" et "end" sont des mots du vocabulaire.
- nomprog, nomvar1, nomvar2 et nomvar3 sont des identificateurs, au même titre que les noms propres de la langue française, on peut donc mettre le nom que l'on veut. Des précisions sont données au 2.3.
- type1 et type2 sont des types parmi ceux autorisés. Voir également au 2.3.
- instruction1 et instruction2 sont des instructions quelconques. Nous en verrons quelques unes dans ce chapitre.

2.3 Types, variables, opérateurs

Un type est un domaine de définition. Il existe notamment les types suivants :
- **integer** : tous les entiers entre -32768 et +32767 (s'ils sont codés sur 16 bits).
- **real** : les réels, codés sur 4 octets comme nous l'avons vu au 1.3, ou parfois sur 8.
- **char** : les caractères du clavier.
- **boolean** : c'est un ensemble formé des 2 éléments suivants {true; false}

Les booléens sont vus au chapitre 3.
- **string** : ce type non standard (tous les Pascal ne disposent pas de ce type) est utile pour représenter les chaînes de caractères. 'bonjour' est une chaîne de caractères.
On verra plus tard le type tableau.

Une variable a en informatique le même sens qu'en mathématique. Plus précisément, une variable a les caractéristiques suivantes :
- un identificateur (autrement dit un nom). On peut donner le nom que l'on veut ou presque : un identificateur doit commencer par une lettre de l'alphabet, suivie ou non de signes alphanumériques. Par exemples, x, total, x1, x234tt, AaBb sont des noms de variable possibles, mais 2x, gh?st, kk** ne le sont pas.
- un type parmi ceux cités plus haut
- une valeur
Attention, pour différencier le nom des variables de type caractère des caractères eux-mêmes, on met chaque caractère entre 2 apostrophes. Ex : 'x' est un caractère, x est une variable.
- une adresse en mémoire (mémoire vive, bien entendu). On ne se servira pas directement des adresses mémoire des variables, mais il est tout de même bon de savoir qu'elles existent, car elles nous permettront d'expliquer certaines notions des derniers chapitres.

Il existe de nombreux opérateurs et fonctions pour tous les types. Nous classons ci-dessous les opérateurs par rapport aux domaines de définition des opérandes et du résultat :

- Integer x Integer --> Integer: +, -, *, div, mod
x div y est le résultat entier de la division de x par y. Par exemple 5 div 2 = 2
x mod y est le résultat du reste de la division entière de x par y. Par exemple, 5 mod 2 = 1

- Real x Real --> Real : +, -, *, / (sachant qu'un integer est aussi un real)

- Real --> Integer : trunc, round[2]
trunc (x) est la valeur entière de x. Exemple, trunc (4.8) = 4
round (x) est la valeur entière approchée de x. Exemple, round (4.8) = 5

- Real --> Real : cos, sin, atan, ln, sqrt, sqr, abs
sqrt (x) est la racine carrée de x (SQuaRe Transpose en Anglais)
sqr (x) est le carré de x

- Char --> Char : succ, pred
succ (x) est le caractère qui suit x dans la table ASCII qui code les carcatères.
pred (x) est le caractère qui précède x " " " " "

- Char --> integer : ord
ord (x) donne le rang du caractère x dans la table ASCII.

- Integer --> Char : chr
chr (x) donne le caractère qui est codé par le nombre x.

- Real x Real --> Boolean : >, <, =, >=, <=, <>

[2]Notez la nécessité des parenthèses autour de l'opérande lorsque la fonction est unaire.

Les opérateurs de comparaison fournissent un résultat nécessairement vrai ou faux, ce qui correspond au type booléen.

- Les opérateurs spécifiques aux booléens sont vus au chapitre 3.

2.4 L'instruction d'affectation

L'instruction d'affectation a pour but d'affecter (de donner) une valeur à une variable. Sa syntaxe est la suivante :

nomvar := expression mathématique

- nomvar est le nom d'une variable.

- L'expression mathématique est quelconque mais son évaluation doit donner une valeur du même type que celui de la variable. Il existe des niveaux de priorité pour les opérateurs, le niveau le plus bas étant le plus fort :

Niveau 1 : tous les opérateurs unaires
Niveau 2 : *, /, div, mod, and
Niveau 3 : +, -, or
Niveau 4 : =, <, >, <=, >=, <>

Des parenthèses peuvent être utilisées pour modifier les priorités d'évaluation.

2.5 Les instructions d'Entrées-Sorties

Pour afficher des caractères à l'écran, il existe deux instructions :

a) **write** (liste d'expressions)
Une liste d'expressions est une ou plusieurs expressions mathématiques séparées par des virgules. Chaque expression est évaluée et son résultat est affiché.[3]

```
Exemple :   x := 5 ;                { On affecte 5 à la variable x, supposée de type integer }
            write ( 'Bonjour chers amis…') ;
            write ( 10, 3*(x-2), '8') ;

--> Ecran : Bonjour chers amis…     10    9 8
```

b) **writeln** (liste d'expressions)
writeln agit exactement comme write, mais ajoute un retour à la ligne après le dernier affichage. Notons que writeln sans paramètre ni parenthèse permet de passer à la ligne suivante.

```
Exemple :   x := 5 ;        { affectation de 5 à la variable x, supposée de type integer }
            writeln ( 'Bonjour chers amis…') ;
            writeln ( 10, 3*(x-2), '8') ;
```

[3]Pour afficher un apostrophe, qui sert de signe de fin de chaîne, il faut en écrire 2 à la suite. Ex: 'L''autre'

--> Ecran[4] : Bonjour chers amis…
10 9 8

Pour que l'utilisateur attribue une valeur à une variable pendant l'exécution du programme, il faut utiliser l'instruction readln. Sa syntaxe est la suivante :

readln (nomvar)

nomvar doit être une variable définie dans la déclaration des variables.
A l'exécution du programme, lorsque l'instruction readln est rencontrée, l'utilisateur doit taper une valeur, puis la touche return (ou entrée). A ce moment là, la valeur tapée est donnée à la variable nomvar et le programme se poursuit.

Nous pouvons maintenant écrire des petits programmes. Exemple :

```
program moyenne ;            { Calcul de la moyenne de 2 nombres. }
var     x, y : integer ;
        moy : real ;
begin
        writeln ('Entrez la valeur du premier nombre') ;      readln ( x ) ;
        writeln ('Entrez la valeur du deuxième nombre');      readln ( y ) ;
        moy := (x + y) / 2 ;
        writeln ('La moyenne des 2 nombres est : ', moy :4:1) ;
end.
```

--> Ecran : Entrez la valeur du premier nombre
5
Entrez la valeur du deuxième nombre
28
La moyenne des 2 nombres est : 16,5

2.6 Exercices

1. Ecrire un programme qui demande à l'utilisateur les coordonnées de 2 points distincts du plan et qui affiche les coordonnées du point milieu.

2. Quel est le résultat à l'écran du programme suivant :

```
program simple;
var     x,y : integer;
begin
        x := 1; y := 2;
```

[4]Pour améliorer l'affichage et réduire l'espace entre les nombres, on peut mettre un format. Ainsi, si on veut que l'affichage des entiers ne prennent que 2 caractères, il suffit de rajouter :2 après l'expression. Si on avait eu des réels, le format comprend 2 paramètres :p1 :p2, le premier pour spécifier le nombre total de caractères et le second pour spécifier le nombre de chiffres après la virgule.

```
        x := x + 1;       y := y + x;
        writeln ( 'y vaut :', y');
end;
```

3. Ecrire un programme qui demande à l'utilisateur une valeur pour U_0, r et n et qui affiche la énième valeur de la suite arithmétique définie par U_0 et $U_{n+1} = U_n + r$. (On rappelle la propriété : $U_n = U_0 + n.r$)

4. Ecrire un programme qui demande à l'utilisateur les valeurs de 2 entiers x,y, qui permute leur valeur et qui les affiche.

5. Application physique : Au temps t0 = 0, un obus est lancé verticalement en l'air à partir d'une plate-forme située à y0 mètres du sol avec une vitesse initiale de v0 mètres par seconde. En prenant la constante de gravitation égale à 9.81, écrire un programme qui demande à l'utilisateur les valeurs de y0, v0, une valeur de temps t et qui affiche la vitesse v ainsi que la hauteur y de l'obus par rapport au sol précisément à l'instant t. (On évitera de prendre des valeurs de t trop grandes pour éviter de trouver des valeurs de y négatives)

6. Ecrire un programme qui demande à l'utilisateur la valeur d'une durée exprimée en secondes et qui affiche sa correspondance en heures minutes secondes. Ex: 3800 s --> 1 heure 3 minutes 20 secondes.

7. Trouvez toutes les erreurs de syntaxe et toutes les erreurs sémantiques dans le programme suivant qui calcule la moyenne exacte de 3 nombres entiers.

```
programme calcule moyenne;
var     x1 : entier       x2 : entier
        x3 : entier
        moy : entier
begin
        writeln (Entrez 3 nombres entiers); readln(x1); readln(x2); readln(x3);
        moy := (x1+x2+x3 / 3);
        writeln('La moyenne est ', 'moy');
end.
```

8. Ecrire un programme qui demande à l'utilisateur un entier plus petit que 8 et qui affiche le nombre binaire correspondant (utilisation de div et mod).

9. Ecrire un programme qui demande à l'utilisateur un nombre entier de 4 chiffres commençant par 1 et composé de 0 et de 1, puis, en supposant ce nombre binaire, qui affiche le nombre en base 10.

2.7 Solutions

```
1. program milieu ;
var      x1, y1, x2, y2, xm, ym : real  ;
begin
        writeln ('Entrez les coordonnées du premier point') ; readln ( x1 ) ; readln ( y1 );
        writeln ('Entrez les coordonnées du deuxième point') ; readln ( x2 ) ; readln ( y2 );
        xm := (x1 + x2) / 2 ;  ym := (y1 + y2) / 2 ;
        writeln ('Les coordonnées du point milieu sont :', xm :5:2, ym:5:2) ;
end.
```

2. Ecran : y vaut 4
Notez que même si y := y+x est sur la même ligne que x := x+1, x := x+1 est exécuté avant.

```
3. program suite ;
var      U0, Un, r : real  ;        n : integer ;
begin
        writeln ('Entrez les valeurs de U0, r et n') ; readln ( U0 ) ; readln ( r ); readln ( n );
        Un := U0 + n*r;          writeln ('La ',n:3,'ième valeur de la suite est :', Un :5:2) ;
end.
```

```
4. program permutation ;
var x, y, stockage : integer ;
begin
        writeln ('Entrez la valeur de x et y'); readln (x); readln(y);
        stockage := x; { On mémorise la valeur de x pour ne pas l'oublier }
        x := y;             { On met la valeur de y dans x }
        y := stockage; { On met dans y l'ancienne (!!) valeur de x }
        writeln(' x vaut maintenant ',x:4,' et y ', y:4);
end.
```

```
5. program physique;
var v0, y0, t, v, y : real ;
begin
        writeln ('Entrez les valeurs de v0 et y0'); readln (v0); readln (y0);
        writeln ('Entrez une valeur pour le temps t'); readln(t);
        v := -9.81*t + v0;       y := -0.5*9.81*sqr(t) + v0*t + y0;
        writeln ('Vitesse : ',v:5:2,'          Altitude : ',y:5:2);
end.
```

```
6. program temps;                { Une des solutions possibles }
var t, h, mn, s : integer;
begin
        writeln('Entrez la valeur d''une durée en secondes'); readln (t);
        h := t div 3600;          mn := (t mod 3600) div 60;    s := t mod 60;
        writeln('Cette durée équivaut à ', h:2,' heures,', mn :2, ' minutes et ',s:2,' secondes.');
end.
```

7. Voici le programme corrigé. Toutes les erreurs sémantiques sont signalées en commentaires.
Notez que le programme n'est pas changé si on insére ou si on supprime des sauts de ligne ou des blancs entre chaque mot (1 minimum).

```
program calculemoyenne;      { Erreur de syntaxe : pas de blanc dans l'identificateur }
var      x1 : integer; x2 : integer; x3 : integer;
         moy : real;      { Erreur sémantique : la moyenne est un réel ! }
begin
         writeln ('Entrez 3 nombres entiers'); readln(x1); readln(x2); readln(x3);
         moy := (x1+x2+x3) / 3;          { Erreur sémantique : attention aux parenthèses }
         writeln('La moyenne est ', moy); { Erreur sémantique : affichage de la valeur de moy
!}
end.
```

```
8. program decimalversbinaire;
var x, a1, a2, a3 : integer ;
begin
         writeln('Entrez un entier inférieur à 8'); readln (x);
         a1 := x div 4;  a2 := (x mod 4) div 2;a3 := x mod 2;
         write ('Le nombre binaire correspondant est : ',a1:1, a2:1, a3:1);
end.
```

```
9. program binaireversdecimal;
var x, resultat : integer;
begin
         writeln('Entrez un entier de 4 chiffres commençant par 1 et composé de 0 et de 1');
         readln (x);
         resultat := 8 + ((x-1000) div 100)*4 + ((x mod 100) div 10)*2 + x mod 10;
         writeln('En base 10, ce nombre est : ',resultat:2);
end.
```

3. L'instruction conditionnelle, l'instruction composée

3.1 Les booléens

Les booléens sont utilisés pour exprimer qu'une condition est vraie ou fausse. Le type booléen est défini par un ensemble constitué de 2 éléments { true; false }. De même que pour les integer ou les real, il existe des opérateurs faisant intervenir les booléens :

• Le and
Le **and** est un opérateur binaire qui nécessite donc deux opérandes. Un **and** entre 2 expressions booléennes a et b permet d'exprimer qu'une condition est respectée (vraie) si et seulement si les conditions définie par a **et** b sont respectées (vraies). De même qu'il existe une table d'addition ou de multiplication pour les entiers, il existe une table du **and** pour les booléens :

AND	**false**	**true**
false	false	false
true	false	true

Exemple :
Hypothèses :
- A est un booléen qui vaut true si Pierre est plus grand que Paul et false sinon
- B est un booléen qui vaut true si Pierre est plus grand que François et false sinon
(A and B) vaut true si Pierre est plus grand que Paul **et** Pierre est plus grand que François, et vaut false dans tous les autres cas, ce qui correspond à notre logique intuitive.

• Le or
Le **or** est également un opérateur binaire. Un **or** entre 2 expressions booléennes a et b permet d'exprimer qu'une condition est respectée (vraie) si et seulement si l'une **ou** l'autre des deux conditions définies par a et b est respectée (vraie). La table du **or** est présentée ci-dessous.

OR	**false**	**true**
false	false	true
true	true	true

Reprenons notre exemple :
(A or B) vaut true si Pierre est plus grand que Paul **ou** Pierre est plus grand que François **ou** les deux, et vaut false sinon, ce qui correspond là encore à notre logique intuitive.

• Le not
Le not est le troisième opérateur booléen. Il exprime tout simplement la valeur contraire d'un booléen. Ainsi, x étant un booléen, si x vaut true, not x vaut false et inversement, si x vaut false, not x vaut true.

Enfin, il ne faut pas oublier les opérateurs de comparaison entre entier ou réels dont le résultat est un booléen. Par exemple (10 > 5) vaut true et (5 = 6) vaut false. On peut bien entendu combiner les opérateurs : si x, y et z sont des réels, l'expression (x > y) and (x > z) est un booléen qui vaut true si et seulement si x est supérieur à y et z.

3.2 L'instruction conditionnelle

La syntaxe de l'instruction conditionnelle est la suivante :

if expression booléenne **then** instruction1 **else** instruction2

if, **then** et **else** sont des mots du langage Pascal.
Cette instruction sert à effectuer un traitement uniquement lorsqu'une condition bien déterminée est satisfaite. Ainsi, si l'expression booléenne est évaluée à vrai, c'est instruction1 (une instruction au choix) qui est effectuée, et dans le cas contraire, c'est à dire si l'expression booléenne est évaluée à false, c'est instruction2 qui est effectuée. Notons que "else instruction2" est facultatif (il arrive qu'il n'y ait aucune action à réaliser).

Exemple, x, y et z étant des variables entières ou réelles quelconques :

```
if (x > y) and (x > z)  then writeln('x est le plus grand')
                        else writeln('x n''est pas le plus grand') { différent de x est le + petit ! }
```

On peut emboîter les instructions conditionnelles. En cas d'ambiguïté, le else se rapporte toujours au if le plus proche.
Exemple :

```
if (10 < 5) then if (5 > 6) then writeln('premier cas')
else writeln('deuxième cas')
```

Rien n'est écrit à l'écran car 10<5 est faux et le else se rapporte au deuxième if !
Présentons ces instructions plus lisiblement :

```
if (10 < 5) then  if (5 > 6) then writeln('premier cas')
                             else writeln('deuxième cas')
```

3.3 L'instruction composée

La syntaxe de l'instruction composée est la suivante :

```
begin
        instruction1 ;
        instruction2 ;
        …               { On peut en mettre autant qu'on le souhaite }
end
```

L'instruction composée sert à grouper des instructions dans un même ensemble. Cela est particulièrement intéressant avec une instruction conditionnelle, notamment dans le cas d'un traitement nécessitant plus d'une instruction. Notez la présentation, fortement conseillée, avec un décalage des instructions par rapport au begin et au end qui sont eux alignés.

Exemple : programme qui calcule et affiche la racine carrée d'un nombre.

```
program racine;
var x,y : real;
begin
        writeln ('Entrez un nombre quelconque'); readln(x);
        if (x >= 0)  then begin
                                y := sqrt (x);
```

```
                writeln('La racine réelle est : ',y:5:2);
            end
        else begin
                y := sqrt (-x);
                writeln('La racine complexe est : ',y:5:2,'i');
            end;
end.
```

3.4 Exercices

1. Ecrire un programme qui demande à l'utilisateur la valeur de 3 variables réelles et qui affiche le maximum. Donnez une version sans utiliser le else et une version avec.

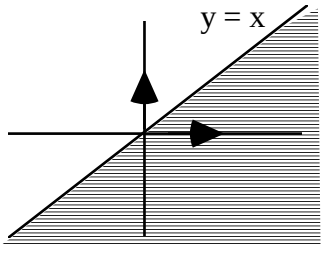
2. Soient 5 variables booléennes A, B, C, D, E exprimant les propositions logiques suivantes :
A : Il faut prendre le chemin de droite
B : Il ne faut pas prendre le chemin de gauche
C : Il ne faut pas prendre le chemin du milieu
D : B dit la vérité et A est un menteur
E : C dit la vérité ou A est un menteur
Sachant que 3 des propositions sont fausses parmi les 5, quel chemin faut-il prendre ?

3. Soient 3 variables booléennes A, B, C exprimant les propositions logiques suivantes :
A : Le livre est à droite de la lampe
B : La lampe est à droite du crayon
C : Le crayon est à gauche du livre
En utilisant les opérateurs and, or et not appliqués à A, B, C, donnez l'équivalence booléenne des propositions suivantes :

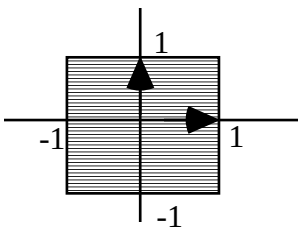
1) Le crayon est l'objet le plus à droite.
2) La lampe est au milieu
3) Le livre n'est pas l'objet le plus à gauche
4) Si le crayon est au milieu, alors la lampe est à droite, sinon la lampe est au milieu.

4. Ecrire un programme qui demande à l'utilisateur les coordonnées (x,y) d'un point P et qui affiche à l'écran si oui on non ce point appartient à la zone hachurée suivante :

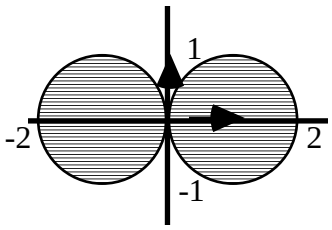
1)



2)



3)



5. Ecrire un programme qui demande à l'utilisateur les valeurs des coefficients a, b, c d'une équation du second degré $a.x^2 + b.x + c = 0$ et qui calcule et affiche les solutions. On traitera également le cas a = 0.

3.5 Solutions

```
1. program maximum1 ;        { Sans utiliser le else }
var x,y, z : real ;
begin
        writeln('Entrez la valeur de 3 réels'); readln(x); readln(y); readln (z);
        if (x >= y) and (x >= z) then writeln('Le maximum est : ', x:5:2);
        if (y > x) and (y >= z) then writeln('Le maximum est : ', y:5:2);
        if (z > x) and (z > y) then writeln('Le maximum est : ', z:5:2);
end.

program maximum2 ; { En utilisant le else }
var x,y, z, max : real ;
begin
        writeln('Entrez la valeur de 3 réels'); readln(x); readln(y); readln (z);
        if (x >= y) then if (x >=z)
                                then writeln('Le maximum est : ', x:5:2)
                                else writeln('Le maximum est : ', z:5:2)
                else if (y > z)
                                then writeln('Le maximum est : ', y:5:2)
                                else writeln('Le maximum est : ', z:5:2);
end.
```

2. Il suffit de prendre les 3 cas possibles et d'évaluer les expressions booléennes. Seul 1 des 3 cas permet d'obtenir 3 propositions fausses et 2 vraies. A vous de trouver la solution.

3. Il faut faire un tableau des cas possibles et déduire l'expression, ou essayer de trouver la solution à l'aide d'un raisonnement logique.

1) E1 := (not B) and (not c)
2) E2 := (A and B) or (not A and not B)
3) E3 := A or C
4) E4 := (B and not C) or E2

4.
```
program hachuree;
var x,y : real;
begin
        writeln('Entrez les coordonnées d''un point du plan.'); readln(x); readln(y);
        if (y <= x) then writeln('Dans la zone hachurée figure 1')
                else writeln('Pas dans la zone hachurée figure 1');
        if (x<=1) and (x >= -1) and (y >=-1) and (y<=1)
                then writeln('Dans la zone hachurée figure 2')
                else writeln('Pas dans la zone hachurée figure 2');
        if (sqr(x-1) + sqr(y) <=1) or (sqr(x+1) + sqr(y) <= 1)
                then writeln('Dans la zone hachurée figure 3')
                else writeln('Pas dans la zone hachurée figure 3');
end.
```

5. Solution non fournie, car pourra faire l'objet d'un T.P.

4. L'instruction itérative for

4.1 L'instruction for

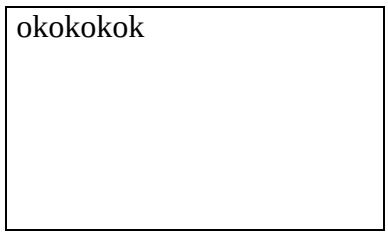
La syntaxe de l'instruction "for" est la suivante :

for nom_de_variable := expression1 **to** expression2 **do** instruction1

Exemple **:**

```
program quatreok;    { Ecrit 4 fois "ok" à l'écran }
var i : integer;
begin
        for i:=1 to 4 do write('ok');
end.
```

Ecran :




```
(* Programme équivalent sans la boucle for *)
program quatreok;    (* Ecrit 4 fois "ok" à l'écran *)
var i : integer;
begin
        i :=1;          write('ok');
        i := i+1;       write('ok');
        i := i+1;       write('ok');
        i := i+1;       write('ok');
end.
```

Principe de fonctionnement :
La boucle for permet d'effectuer des itérations. La variable de boucle "nom_de_variable" prenant tour à tour toutes les valeurs comprises entre valeur1 et valeur2 (valeur1 et valeur2 incluses) définies respectivement par les valeurs de expression1 et expression2, l'instruction "instruction1" est exécutée autant de fois. Ainsi, "nom_de_variable" est initialisée avec valeur1, puis "instruction1" est exécutée, puis "nom_de_variable" prend la valeur qui suit logiquement valeur1 et "instruction1" est à nouveau exécutée etc..

Conditions d'exécution :
- La variable de boucle doit être de type énumérable. En effet, il est nécessaire d'avoir un nombre fini de valeurs comprises entre valeur1 et valeur2. La variable peut donc être de type integer, char ou boolean mais pas real.
- La valeur de la variable de boucle ne doit pas être modifiée dans instruction1. En particulier, il ne faut pas écrire l'affectation de la valeur suivante, celle-ci se fait automatiquement.
- expression1 et expression2 sont des expressions mathématiques quelconques du même type que celui de la variable de boucle.
- valeur1 doit précéder ou être égal à valeur2. Toutefois, si la condition n'est pas respectée, il n'y a pas d'erreur de syntaxe et l'instruction1 n'est pas exécutée.

- instruction1 peut être n'importe quelle instruction, y compris un autre for. Si plusieurs instructions doivent être exécutées dans la boucle, il faut les regrouper à l'intérieur d'une instruction composée begin … end.
- En sortie de boucle, la valeur de la variable de boucle est valeur2.

Variante :

for nom_de_variable := expression1 **downto** expression2 **do** instruction1

Dans cette variante avec "downto", le principe d'itération est le même, mais la variable de boucle prend tour à tour des valeurs décroissantes.

Exemple :
```
program compterebours;      { Affiche les nombres de 10 à 5 }
var     i : integer;
begin
        for i := 10 downto 6 do writeln (i);
end.
```

Ecran :

```
10
9
8
7
6
```

4.2 Exercices

1. Ecrire un programme qui demande une valeur de n à l'utilisateur et affiche les n premiers nombres pairs par ordre croissant, en commençant par 0.

2. Ecrire un programme qui demande une valeur de n à l'utilisateur et affiche les n premiers nombres impairs par ordre décroissant.

3. Soit la suite définie par $\{U_1 = 10; U_{n+1}= 2.U_n - 3\}$. Ecrire un programme qui demande une valeur de n à l'utilisateur et qui affiche les n premiers membres de cette suite.

4. Ecrire un programme qui demande à l'utilisateur une valeur pour deux entiers n1 et n2 et qui affiche le résultat de la multiplication de n1 par n2 sans utiliser le signe de la multiplication.

5. Ecrire un programme qui demande à l'utilisateur une valeur de n et qui affiche le résultat de n!.

6. Ecrire un programme qui demande à l'utilisateur une valeur pour un réel x et un entier y positif et qui affiche le résultat de x^y.

7. Ecrire un programme qui demande à l'utilisateur une valeur de n puis calcule la somme des n premiers carrés ($1 + 4 + 9 + 16 + 25\ldots + n^2$) en affichant tous les résultats intermédiaires.

8. Ecrire un programme qui demande à l'utilisateur une valeur pour deux entiers x et y et qui affiche les motifs suivants en utilisant judicieusement le write et le writeln :

a)
```
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
```
y lignes de x étoiles

b)
```
*
* *
* * *
* * * *
```
y lignes avec 1 étoile de plus à chaque nouvelle ligne
(x n'est pas utilisé)

c)
```
* * * * * * *
* * * * * *
* * * * *
* * * *
```
y lignes avec x étoiles à la première ligne
et 1 étoile de moins à chaque nouvelle ligne

d) Un cercle plein d'étoiles collé au bord gauche de l'écran avec un rayon de x étoiles (la forme est approchée).

```
       *******
    ***************
  ***********************
 ***********************
  ***********************
    ***************
       *******
```

9. Ecrire un programme incluant 1 seul write et 1 seul writeln qui affiche le motif suivant :

```
*
**
****
********
```

10. Ecrire un programme qui affiche les 10 premières lettres de l'alphabet en utilisant un for.

4.3 Solutions

```
1. program pairs;       { Une des solutions possibles }
var i, n : integer;
begin
        writeln('Combien de nombre pairs voulez-vous afficher ?'); readln(n);
        for i:=1 to n do writeln( 2*i - 2 );      { i varie de 1 à n car on doit afficher n nombres }
end.                                              { 2*i-2 se déduit logiquement }

2. program impairs;    { Une des solutions possibles }
var i, n : integer;
begin
        writeln('Combien de nombre impairs voulez-vous afficher ?'); readln(n);
```

```
        for i:=1 to n do writeln( 2*(n-i) + 1); { i varie de 1 à n car on doit afficher n nombres }
end.                                            { 2*(n-i)+1 se déduit logiquement }
```

3.
```
program suite;
var Un, i, n : integer;  { Un va représenter tour à tour chacun des membres de la suite }
begin
        writeln('Combien de nombres de la suite voulez-vous afficher ?'); readln(n);
        Un := 10;
        for i:=1 to n do          { i varie de 1 à n car on doit afficher n nombres }
        begin
                writeln ('Le numéro ',i:2, ' est : ', Un:4);
                Un := 2*Un - 3;         { La nouvelle valeur de Un est égale à 2 fois l'ancienne }
        end;                            { moins 3. Cette façon de programmer est classique. }
end.
```

4.
```
program multiplication;     { Multiplication de 2 entiers positifs sans le signe * }
var i, n1, n2, res : integer;      { res est l'abréviation de résultat }
begin
        writeln ('Entrez la valeur de 2 entiers'); readln(n1); readln(n2);
        res := 0;
        for i:=1 to n2 do res := res + n1;
        writeln('Le résultat de la multiplication est : ', res:4);
end.
```

5.
```
program factorielle; var i,n, res : integer;  { Illustration d'une présentation peu lisible }
begin
        writeln('Entrez un nombre n'); readln(n); res := 1; for i:=2 to n do res := res * i;
        writeln('La factorielle de ce nombre est : ', res:5);
end.    { Attention, si n est grand, il faut définir res en real car la limite de integer est 32767. }
```

6.
```
program puissance;
var x, res : real;          i,y : integer;
begin
        writeln('Entrez un nombre réel, puis la valeur d''un exposant'); readln(x); readln(y);
        res := 1;
        for i:=1 to y do res := res * x;
        writeln('Le résultat est : ', res:5:2);
end.
```

7.
```
program sommecarres;
var i,n, res : integer;
begin
        writeln('Entrez la valeur de n'); readln(n);
        res := 0;
        for i:=1 to n do
        begin
                res := res + i*i;
                writeln('La somme des ',i:2,' premiers carres est : ',res:4);
        end;
end.
```

8. Solution non fournie. Pourra faire l'objet d'un T.P.

9.
```
program affichage;
var     ligne, colonne, nbcol : integer;
begin
        nbcol := 1;
        for ligne := 1 to 4 do
        begin
                for colonne := 1 to nbcol do
                        write('*');
                writeln;
                nbcol := 2*nbcol;
        end;
end.
```

10.
```
program alphabet;
var x : char;
begin
        for x:= 'a' to 'j' do write( x );
end.
```

5 L'instruction itérative while

5.1 L'instruction while

Syntaxe :

1 2

while (expression booléenne) **do** instruction

Tant que l'expression booléenne (1) est évalué à true, l'instruction (2) est exécutée.

Exemple :
Demande d'un nombre à l'utilisateur et vérification qu'il respecte les conditions imposées :

• Solution 1 :

```
writeln ('Tapez au clavier un nombre compris entre 1 et 100'); readln ( N );
while ( (N<1) or (N>100) ) do
begin
        writeln ('Donnez-moi un autre nombre ! '); readln ( N );
end;
```

• Solution 2 :

```
continuons := true;     { continuons est déclaré en variable de type booléen }
while (continuons) do
begin
        writeln ('Tapez au clavier un nombre compris entre 1 et 100'); readln ( N );
        if (N>=1) and (N<=100) then continuons := false;
end;
```

5.2 Comparaisons entre le for et le while

L'instruction for peut toujours être traduite par un while. Exemple :

```
program factorielle1;
var     i, n: integer;              fact : real;
begin
        writeln('Entrez un nombre entier'); readln (n);
        fact := 1;
        for i:=2 to n do
                fact := fact * i;
        writeln ( n,'! =', fact :8:0);
end.
```

```
program factorielle2;
var     i, n : integer;  fact : real; { fact est déclaré en real, car les integer sont < 32768 }
begin
        writeln('Entrez un nombre entier'); readln (n);
        fact := 1;
        i:=2;
        while (i <= n) do
```

```
        begin
                fact := fact * i;
                i := i + 1;
        end;
        writeln ( n,'! =', fact:8:0);
end.
```

Cas général :

```
        for i:=valeur1 to valeur2 do instruction
                        est équivalent à :
        i := valeur1;
        while ( i<= valeur2) do
        begin
                instruction;
                i := i+1;
        end;
        i := i-1;          { car la valeur en sortie doit être valeur2 }
```

Conseils d'utilisation :

- On utilise un for lorsqu'on sait combien de fois on doit passer dans la boucle.
- Pour le while, s'assurer qu'il n'y a pas de boucle infinie : l'expression booléenne doit un jour devenir fausse !
- Bien identifier le traitement qui doit être répété.
- Vérifier les conditions d'entrée dans la boucle, les premières valeurs des variables.
- Vérifier les conditions de sortie de la boucle, les dernières valeurs des variables.

5.3 Exercices

1. Montrez l'évolution du contenu des variables x et y, puis déduire le résultat à l'écran du programme suivant :

```
program monwhile;
var     x,y : integer;
begin
        x := 0; y :=1;
        while (x <=2) do
        begin
                x := x + 1;       y := y + x;
        end;
        writeln ('La valeur finale de y est : ', y);
end;
```

2. Un professeur a corrigé ses copies et veut calculer la moyenne des notes. Ecrire un programme qui demande les notes une par une, qui s'arrête lorsque le professeur rentre la valeur -1 (ce qui signifie qu'il n'y a plus de note) et qui affiche la moyenne des notes.

3. Recherche dichotomique. Calculer l'abscisse du point d'intersection entre les courbes y= ln(x) et y=1/x avec une précision de 0,001. La méthode consiste à choisir 2 points d'abscisses x1, x2 autour de l'intersection et à situer le point milieu. Selon sa position, on l'échange avec

un des 2 points ce qui permet de diviser l'intervalle par 2. Ensuite on recommence. On pourra exploiter le fait qu'à gauche de l'intersection $\ln(x)<1/x$ et qu'à droite, au contraire, $\ln(x)>1/x$.

4. Jeu du devine le nombre : écrire un programme qui crée un nombre entier aléatoire x entre 0 et 9999 et qui demande à un joueur humain d'essayer de le trouver. A chaque essai du joueur, l'ordinateur doit répondre si celui-ci est "trop grand", "trop petit" ou s'il a "gagné". Le jeu se poursuit tant que l'être humain n'a pas trouvé le nombre x. Lorsque le joueur a gagné, l'ordinateur annonce le nombre d'essais effectués par le joueur. L'ordinateur demande ensuite au joueur s'il veut rejouer et agit en conséquence.
Il existe en Pascal un générateur de nombres aléatoires. Pour l'utiliser, il faut écrire l'instruction randomize au début du programme (après le begin). Ensuite, la fonction random(expression) a pour valeur un nombre réel compris entre 0 et la valeur de l'expression.
Exemple :

```
x := random (100);          { Donne à x une valeur entre 0 et 100, plus exactement [0,100[ }
```

5. Ecrire un programme qui demande à l'utilisateur un nombre entier positif et détermine si celui-ci est un nombre premier, en utilisant un while.

6. Jeu des allumettes. Ce jeu se joue à 2. On dispose au départ de n allumettes sur un tapis. Chacun son tour, un joueur prend une, deux ou trois allumettes. Le joueur qui prend la dernière allumette a perdu.
Version 1 : Ecrire un programme qui permet à 2 personnes de jouer à ce jeu. Le nombre d'allumettes au départ est demandé à l'utilisateur.
Version 2 : Le joueur va jouer contre l'ordinateur. La stratégie de l'ordinateur est simple : s'il reste k allumettes avec $2 \leq k \leq 4$, l'ordinateur en prend k-1 pour gagner, sinon il en prend au hasard 1, 2, ou 3.
Version 3 : S'il le peut, l'ordinateur laisse sur le tapis un multiple de 4 plus 1 allumettes.

5.4 Solutions

1. Evolution des variables x et y.

x	0	1	2	3
y	1	2	4	7

Affichage à l'écran : La valeur finale de y est : 7

2.
```
program moyenne; { On suppose qu'il y a au moins 1 note }
var x, somme : real;   n : integer;      continuer : boolean;
begin
        n := 0;          continuer := true;
        while continuer do
        begin
                writeln('Entrez une note'); readln( x );
                if (x <> -1) then begin
                                        n := n+1; somme := somme + x;
                                  end
                             else continuer := false;
         end;
         writeln('La moyenne est : ', somme / n :5:2);
end.
```

3. program dichotomie;
```
var erreur, x1, x2, xmilieu : real;
begin
        x1 := 1; x2 := 3;       { L'abscisse de l'intersection est entre 1 et 3 }
        erreur := 2;
        while (erreur > 0.001) do
        begin
                xmilieu := (x1+x2) / 2;
                if ( ln (xmilieu) > 1/xmilieu )
                then x2 := xmilieu    { Le milieu est à droite de l'intersection }
                else x1 := xmilieu;   { Le milieu est à gauche de l'intersection }
                erreur := abs( x2-x1);
        end;
        writeln ('L''abscisse de l''intersection est, à 0.001 près : ', xmilieu :5:3);
end.
```

4. Solution non fournie. Pourra faire l'objet d'un T.P..

5. program nombrepremier;
```
var i, n : integer;             estpremier, fini : boolean;
begin
        writeln ('Entrez un nombre entier'); readln(n);         { n doit être inférieur à 32767 }
        estpremier := true;
        fini := false;
        i := 2;
        while (not fini) do
        begin
                if (n mod i) = 0 then begin
                                                estpremier := false;
                                                fini := true;
                                        end
                                else begin
                                                i := i+1;
                                                if (i > n div 2) then fini := true;
                                        end;   { Il est inutile de chercher des diviseurs > n div 2 }
        end;
        if (estpremier) then writeln('C''est un nombre premier')
        else writeln('Ce n''est pas un nombre premier, car il est divisible par ',i:5);
end.
```

6. Solution non fournie.

6 Les tableaux unidimensionnels

6.1 Déclaration d'une variable de type tableau

Un tableau est un type au même titre que integer ou boolean, nous pouvons donc avoir des variables qui sont de type tableau. La syntaxe de la déclaration d'un tableau est la suivante :

nom_de_variable : **array** [i1..i2] **of** nom_de_type;

- i1 et i2 sont deux valeurs de type énumérable, en général des entiers, avec i1 < i2. Il est possible de donner à i1 ou i2 le nom d'une constante définie en tant que tel dans le programme. Les constantes se dÈclarent au début du programme avec const. Voir l'exemple du 6.3.
- nom_de_type est un type parmi real, integer, char, boolean, ou même array, ce dernier cas n'étant pas traité ici.

Un tableau est en fait un ensemble de cases où sont stockées les valeurs de variables de type "nom_de_type". Chaque case du tableau est repérée par un indice et possède une valeur qui lui est propre. Par exemple, ci-dessous, nous avons un tableau de réels de 10 cases, indicées de 1 à 10, qui aurait pu être déclaré en variable de la façon suivante :

tab : array[1..10] of real; (* tab est le nom donné à notre tableau *)

1	2	3	4	5	6	7	8	9	10
43.2	25.0	31.3	32.1	34.2	25.7	32.0	32.5	44.0	32.5

6.2 Lecture ou écriture dans un tableau

Pour accéder à une valeur du tableau,, il suffit d'écrire : nom_de_tableau [indice]

L'indice peut être n'importe quelle expression mathématique de type énumérable cohérent avec celui de la déclaration. Exemple, en reprenant le tableau ci-dessus :

```
x := tab[4] + 1;          (* on affecte à x la valeur de la case n°4 + 1, soit 33.1 *)
tab[k+1] := 0; (* k étant un entier entre 0 et 9, on met 0 dans la case n°(k+1) *)
```

6.3 Exemples de programme avec utilisation d'un tableau

```
program tableau1;     (* Ce programme calcule la moyenne des valeurs d'un tableau *)
const n = 10;  (* n vaut 10 dans tout le programme et sa valeur ne peut changer *)
var     i , somme : integer;
        tab : array [ 1..n ] of integer;          (* tab est un tableau de n entiers *)
begin
        (*  Demande des n valeurs du tableau à l'utilisateur *)
        for i:=1 to n do
        begin
                writeln ('Entrez la valeur de la ',i,'ème case'); readln ( tab[i] );
```

```
        end;
        (* Calcul de la somme des n valeurs *)
        somme := tab[ 1 ];      (* La somme est initialisée avec la 1ère valeur du tableau *)
        for i:=2 to n do somme := somme + tab[i];
        writeln ('La moyenne des ',n:2,' valeurs est ', somme / n);
end.

program tableau2;   (* On sait que la suite {U1 = 0; Un+1 = Un / 2 + 1 } converge      *)
                    (* vers 2 par valeur inférieure. Ce programme stocke les 100      *)
                    (* premiers membres de la suite dans un tableau, puis demande     *)
                    (* la valeur d'un réel delta à l'utilisateur et affiche l'indice du 1er  *)
var                 (* élément du tableau qui approche la valeur 2 à delta près.*)
        U : array[ 1..100 ] of real;
        i : integer;
        delta : real;
begin
        (* Calcul et stockage des 100 premiers membres de la suite *)
        U[1] := 0;
        for i:=2 to 100 do U[ i ] := U [i-1] / 2 + 1;

        (* Demande de la précision désirée à l'utilisateur *)
        writeln ('Quelle précision désirez-vous ?');
        readln ( delta );

        (* Recherche du premier élément approchant la valeur 2 à delta près. *)
        (* (on suppose que cet élément existe parmi les 100 stockés) *)
        i := 1;
        while ( 2 - U[i] > delta) do i := i + 1;

        writeln ('Le premier élément approchant 2 avec cette précision est le ',i,'ème');
end.
```

6.4 Déclaration d'un type tableau

Si l'on manipule souvent un même type de tableau, il est fastidieux d'écrire à chaque fois la même déclaration de variable. Pour simplifier et clarifier cette déclaration, il est possible de créer un type tableau et d'assigner ensuite ce type aux variables tableau correspondantes.

```
type                        (* Déclaration des types *)
        nouveau_type1 = array[ liste d'intervalles ] of nom_de_type1;
        nouveau_type2 = array[ liste d'intervalles ] of nom_de_type2;
        …
var                         (* Déclaration des variables *)
        nom_tab1, nom_tab2 : nouveau_type1;
        nom_tab3 : nouveau_type2;
        …
```

Exemple :

```
const   dimension = 3;          (* On travaille dans un espace a 3 dimensions *)
                                (* Pour changer de dimension, il suffit de changer la constante *)
type    vecteur = array[1..dimension] of real;
var     v1, v2, v3 : vecteur;
```

6.5 Exercices

1. Ecrire un programme qui demande à l'utilisateur 10 valeurs réelles, qui les stocke dans un tableau et qui affiche la plus grande valeur ainsi que son rang dans le tableau.

2. Ecrire un programme qui demande à l'utilisateur 10 valeurs entières correspondant à des notes entre 0 et 20, qui les stocke dans un tableau et qui affiche combien de notes sont supérieures ou égales à 10.

3. Soit T un tableau de 7 caractères correspondant à un mot. Ecrire les instructions permettant de modifier T de sorte que le mot soit renversé. Par exemple "bonjour" devient "ruojnob".

4. Soit T un tableau de 10000 caractères (array[1..10000] of char) correspondant au texte d'un document quelconque. Pour corriger le texte, l'utilisateur voudrait remplacer un mot de 5 lettres par un autre mot de 5 lettres. Ecrire la suite d'instructions qui permet de demander à l'utilisateur les 2 mots, de trouver toutes les occurrences du premier mot et de les remplacer par le deuxième.

5. Ecrire un programme efficace qui trouve et affiche tous les nombres premiers inférieurs à 10000.

6. On désire effectuer des calculs sur les précipitations relevées chaque mois dans un même endroit. On propose pour cela la structure de programme suivante :

```
program ilpleutbeaucoup;
var     mois : array[1..12] of string;  { string est utilisé pour les chaînes de caractères }
        pluies : array[1..12] of integer; { Précipitations en mm pour chaque mois }
begin
        mois[1] := 'janvier';           { Compléter avec les autres mois }
        { A compléter en répondant dans l'ordre aux questions b, c, d }
        { On n'oubliera pas de déclarer d'éventuelles nouvelles variables }
end;
```

a) Compléter l'initialisation du tableau mois au début du programme.
b) Ecrire les instructions permettant le remplissage du tableau pluies par l'utilisateur.
c) Ecrire les instructions permettant d'afficher l'écart de précipitation entre le mois le plus pluvieux et le mois le moins pluvieux ainsi que le nom de ces mois.
d) Ecrire les instructions permettant d'afficher l'histogramme des précipitations en utilisant le symbole '*' (étoile), avec en ordonnée le numéro du mois et en abscisse les précipitations, normalisées pour avoir entre 1 et 10 étoiles, 10 pour le mois le plus pluvieux.
e) Modifier d) pour que l'histogramme apparaisse verticalement.

6.6 Solutions

1. program maxtableau;
```
var     i , rang : integer;                 max : real;
        tab : array [ 1..10 ] of real;
begin
        for i:=1 to 10 do       (*  Demande des 10 valeurs du tableau à l'utilisateur *)
        begin
                writeln ('Entrez la valeur numéro ',i:2); readln ( tab[i] );
        end;
        max := tab[1]; rang := 1;
        for i:=2 to 10 do
                if tab[i] > max then begin
                                        max := tab[i]; rang := i;
                                    end;
        writeln('Le maximum est ',max:6:2,' en position ',rang:2);
end.
```

2. program nombredemoyennes;
```
var     i , n : integer;
        tab : array [ 1..10 ] of integer;
begin
        for i:=1 to 10 do begin writeln ('Entrez la note numéro ',i:2); readln ( tab[i] ); end;
        n := 0;
        for i:=1 to 10 do
                 if tab[i] >= 10 then n := n+1;
        writeln('Il y a ',n:2,' notes supérieures ou égales à la moyenne');
end.
```

3. On demande "les instructions", pas tout le programme, donc on suppose que T est déjà rempli avec des caractères. Dans la suite, on utilise deux variables : i de type integer et memoire de type char.
```
for i:=1 to 3 do { On va effectuer 3 permutations pour obtenir le mot symétrique }
begin
        memoire := t[i]; { Pour permuter les valeurs de 2 variables, il en faut une 3ème ! }
        t[i] := t[8-i];     { i croît de 1 à 1 à partir de 1, 8-i décroît de 1 en 1 à partir de 7 }
        t[8-i] := memoire;  { t[8-i] reçoit l'ancienne valeur de t[i] mémorisée dans "memoire" }
end;
```

4. On suppose là encore que le tableau de caractères est déjà rempli. On ajoute dans la déclaration des variables 2 tableaux de 5 caractères m1 et m2 pour représenter les 2 mots, 2 integer i et j ainsi qu'un booléen appelé trouve.
```
m1, m2 : array[1..5] of char;
Instructions demandées :
        writeln('Entrez les lettres du mot à changer une par une');
        for i:=1 to 5 do readln(m1[i]);
        writeln('Entrez les lettres du mot remplaçant une par une');
        for i:=1 to 5 do readln(m2[i]);
        i := 1;
        while (i <= 9996) do         { 9996 car de 9997 à 10000, il ne reste que 4 cases }
        begin
                 if (T[i] = m1[1])
```

```
                then begin          { La première lettre a été trouvée, il faut vérifier les autres }
                        trouve := true;
                        for j:=1 to 4 do        { Si 1 lettre diffère, le mot n'est pas trouvé }
                          if (T[i+j] <> m1[j+1]) then trouve := false;
                        if (trouve)
                        then begin   { On remplace les 5 lettres trouvées dans T par m2 }
                                for j:=1 to 5 do T[i-1+j] := m2[j];
                                i := i+4;  { On avance de 4 cases + 1 ci-dessous, ça fera 5}
                        end;
                i := i + 1;             { On avance d'une lettre à chaque fois }
        end;
```

5.
```
program nombrespremiers;          (* Recherche des nombres premiers inférieurs à 10000 *)
var     t : array[1..10000] of boolean;         (* Les nombres premiers seront marqués true *)
        i, n : integer;             (* compteurs des nombres entre 1 et 10000 *)
        multiple : integer;       (* prend tour à tour les multiples successifs d'un nbre 1er *)

begin
        (* Initialisation à true du tableau des nombres premiers *)
        for i:=1 to 10000 do t[i] := true;

        (* On va prendre les nombres premiers 1 à 1 et éliminer tous ses multiples *)
        (* 1 est un nombre premier à part, car tous les nombres sont multiples de 1, donc *)
        (* on commence à 2 *)
        for n:=2 to 10000 do
        begin
                if (t[n] = true) (* On vient de trouver un nombre premier *)
                then begin
                        multiple := n;  (* On élimine tous les multiples de n *)
                        while (multiple+n <= 10000) do
                        begin
                                multiple := multiple + n;
                                t[multiple] := false;
                        end;
                    end;
        end;
        (* Affichage des nombres premiers *)
        for i:=1 to 10000 do
                if (t[i] ) then writeln ( i : 5,' est un nombre premier');
end.
```

6. Solution non fournie. Pourra faire l'objet d'un T.P.

7 Les procédures et les fonctions

7.1 Les fonctions

Il existe des fonctions prédéfinies simples comme sqrt(x), ou cos(x). Pascal autorise la construction d'autres fonctions plus complexes comprenant un ou plusieurs paramètres. La déclaration d'une fonction se justifie lorsqu'un même traitement particulier ayant pour but de calculer une valeur à partir d'un ensemble de données se répète plusieurs fois dans le programme. Par exemple, on peut avoir besoin de calculer la distance entre 2 points à des étapes différentes du programme. Pour ne pas écrire plusieurs fois la formule, il est possible de déclarer une fonction qui décrit ce calcul et de l'appeler précisément lorsqu'on en a besoin.

• Déclaration d'une fonction
La déclaration d'une fonction s'écrit toujours avant le corps principal du programme et avant toutes les autres procédures et fonctions qui l'utilisent. La syntaxe de la déclaration est la suivante :

```
function nomdefonction ( déclaration des paramètres ) : type de la fonction;
var     (* Déclaration d'éventuelles variables locales *)
        nomvar1 : type1;
        nomvar2 : type2;
        …
begin
        (* Corps de la fonction *)
        instruction1;
        instruction2;
        …
        nomdefonction := expression;          (* Affectation de la valeur finale *)
end;
```

- Syntaxe de la déclaration des paramètres : (nomvar1 : type1 ; nomvar2 : type2 …)
- Attention, le type d'un paramètre ou celui de la fonction ne peut pas être array.
- Comme leur nom l'indique, les variables "locales" ne peuvent servir qu'à un calcul local, à l'intérieur de la fonction, elles ne sont donc pas connues ailleurs ! Grâce à cette propriété, on peut déclarer une variable locale avec un nom déjà utilisé pour une variable d'une autre fonction. Celles-ci sont indépendantes l'une de l'autre et ne sont accessibles qu'à l'intérieur de la fonction où elles ont été declarées. Attention, les variables déclarées au début du programme sont dites globales, ce qui signifie qu'elles sont connues dans tout le programme. En cas d'ambiguïté avec une variable locale, c'est la variable locale qui est reconnue.
- Les paramètres d'une fonction ont les mêmes propriétés que les variables locales.

• Appel d'une fonction
Toute fonction qui a été déclarée peut être utilisée dans une expression. Il suffit d'écrire son nom suivi d'une liste de "paramètres d'appel" entre parenthèses séparés par des virgules. Pour que l'appel de la fonction soit correcte, il doit y avoir autant de paramètres d'appel que de paramètres déclarés et le type de chacun d'eux doit être rigoureusement identique.

Exemple :

```
program equilateral; { Ce programme teste si un triangle est équilatéral, voir aussi le 8.1 }
var     cote1, cote2, cote3 : real;
        x1,y1, x2, y2, x3, y3 : real;

{ ********* Début de la déclaration de la fonction distance ************ }
function distance (x1 : real; y1 : real; x2 : real; y2 : real ) : real;
{ Pas besoin de variable locale }
begin
        distance := sqrt ( sqr(x2-x1) + sqr(y2-y1) );
end;
{ ************** fin de la déclaration de la fonction ***************}

begin  (* Corps principal du programme. *)
        writeln ('Entrez les coordonnées des 3 points du triangle ');
        readln(x1); readln(y1); readln(x2); readln(y2); readln(x3); readln(y3);
        cote1 := distance (x1, y1, x2, y2);             { Appel de la fonction distance }
        cote2 := distance (x2, y2, x3, y3);             { Appel de la fonction distance }
        cote3 := distance (x3, y3, x1, y1);             { Appel de la fonction distance }
        if (cote1 = cote2) and (cote2 = cote3)
                then writeln ('Le triangle est équilatéral')
                else writeln ('Le triangle n''est pas équilatéral');
end.
```

7.2 Les procédures

Comme pour une fonction, l'intérêt d'une procédure est de pouvoir regrouper les instructions complexes d'un traitement et de lui donner un nom. Toutefois, une fonction renvoie une valeur, alors qu'une procédure ne le fait pas. Une procédure joue en fait le rôle d'une méta-instruction regroupant plusieurs instructions. Par exemple, on peut avoir besoin d'afficher un motif similaire à différents endroits du programme. Il suffit alors de regrouper les instructions d'affichage au sein d'une même procédure, éventuellement paramétrée pour tenir compte de cas particuliers, et d'appeler la procédure à chaque fois qu'il est nécessaire d'afficher.

Une procédure se déclare comme une fonction, sauf qu'elle n'a pas de type, ne renvoie pas de valeur et la déclaration commence par "procedure" suivi de l'identificateur. Comme pour une fonction, une procédure doit être déclarée avant le bloc principal et avant toute procédure ou fonction y faisant référence.

Exemple de programme comportant une procédure :

```
program facto;          (* Calcul des 10 premières factorielles *)
var     i : integer;
procedure ecritfactorielle ( n : integer );     (* Affiche la valeur de n!  *)
var     i : integer;    (* i existe aussi en global; c'est le i local qui prime *)
        fact : real;    (* on prend un real car integer est limité à 32767 *)
begin
        fact := 1;
        for i:=1 to n do fact := fact*i;
        writeln ('La factorielle de ', n:2, ' est ', fact:8:0 );
end;
```

```
begin  (* Le programme commence ici *)
   for i:=1 to 10 do            (* Notez que la valeur du i local à la procédure est inconnue *)
     ecritfactorielle ( i );
end.
```

7.3 Exercices

1. Ecrire un programme qui calcule les 10 premières factorielles en utilisant une fonction.

2. Soit montypetab un nouveau type défini de la façon suivante :
type montypetab = array[1..100] of integer;
Soit tab une variable globale de type montypetab.
a) Ecrire une procédure sans paramètre qui demande à l'utilisateur la valeur de toutes les cases du tableau.
b) Ecrire une fonction qui prend comme paramètre un entier n et qui compte le nombre de valeurs du tableau égales à n.
c) Ecrire une fonction sans paramètre qui calcule la somme des valeurs du tableau.
d) Ecrire une fonction sans paramètre qui, en 1 instruction, calcule la moyenne des valeurs du tableau.
e) Ecrire le bloc principal du programme pour que l'utilisateur soit amené à remplir le tableau, et que soit affiché ensuite le nombre de valeurs égales à 0 ainsi que la moyenne des valeurs. On utilisera au mieux les procédures et fonctions déjà écrites.

3. Variante du master-mind.
Cette variante du master-mind consiste a trouver une combinaison de 5 chiffres **sans double**, pris dans l'ensemble des chiffres de 0 à 9. Dans notre cas, c'est l'ordinateur qui va choisir cette combinaison et c'est l'utilisateur qui va tenter de la découvrir. A chaque essai de 5 chiffres, l'ordinateur doit répondre combien de chiffres sont bien placés et combien de chiffres sont mal placés. Lorsque le joueur a trouvé, l'ordinateur doit afficher le nombre d'essais.
Pour réaliser ce programme, on utilisera les structures de données suivantes :

```
type    tab5 = array[1..5] of integer;
var     tabsecret : tab5; { Tableau de référence, rempli aléatoirement mais sans doublon }
        tabessai : tab5;          { Tableau rempli pat l'utilisateur à chaque essai }
```

On définira et utilisera les fonctions et procédures suivantes :

```
procedure inittabsecret;      { initialisation aléatoire (voir exercice 4 du 5.3) de tabsecret }
procedure demandeessai;       { saisie de l'essai du joueur dans tabessai }
function nbbienplaces ( reference : tab5; essai : tab5 ) : integer;
function nbmalplaces ( reference : tab; essai : tab; bienplaces : integer ) : integer;
```

4. On considère le programme incomplet suivant :

```
program triangle;
const   minx = 0;       maxx = 20;
        miny = 0;       maxy = 10;
type    tabpoint = array[1..2] of real;          { pour les 2 coordonnées d'un point }
var     p1, p2, p3 : tabpoint ;
        numeropoint : integer;
        propriete : string;
begin
```

```
        writeln('Entrez les coordonnées du premier point');
        p1[1] := demandevaleur (minx, maxx);        p1[2] := demandevaleur(miny, maxy);
        writeln('Entrez les coordonnées du deuxième point');
        p2[1] := demandevaleur (minx, maxx);        p2[2] := demandevaleur(miny, maxy);
        writeln('Entrez les coordonnées du troisième point');
        p3[1] := demandevaleur (minx, maxx);        p3[2] := demandevaleur(miny, maxy);
        propriete := 'quelconque';
        if (estisocele ( p1, p2, p3 ) then propriete := 'isocèle';
        if (estequilateral (p1, p2, p3) then propriete := 'équilatéral';
        affichetriangle ( propriete );
end.
```

a) Ecrire la fonction demandevaleur qui demande la valeur d'une coordonnée à l'utilisateur et vérifie que celle-ci est comprise dans l'intervalle dont les bornes sont données par les paramètres de la fonction.
b) Ecrire la fonction booléenne estisocele qui prend la valeur true si et seulement si le triangle formé par les 3 points passÈs en paramËtres est isocèle. Il est sans doute opportun d'écrire une autre fonction qui calcule la distance entre 2 points.
c) Ecrire la fonction booléenne estequilateral qui prend la valeur true si et seulement si le triangle formé par les 3 points passés en paramètres est équilatéral.
d) Ecrire la procédure affichetriangle qui affiche à l'écran un triangle composé d'*, respectant la convention suivante :
- il y a 1 seule * sur la première ligne.
- il y a toujours 1 * dans la première colonne de chaque ligne.
- il y a toujours 7 lignes.
- si le triangle est quelconque, il y a cinq * de plus à chaque ligne.
- si le triangle est isocèle, il y a deux * de plus à chaque ligne.
- si le triangle est équilatéral, il y a 4 étoiles de plus à chaque ligne jusqu'à la ligne 4, puis 4 étoiles de moins à chaque ligne jusqu'à la dernière ligne.

7.4 Solutions

```
1. program facto;       (* Calcul des 10 premières factorielles *)
var i : integer;
function factorielle ( n : integer ) : real;
var     i : integer;      fact : real;
begin
        fact := 1;
        for i:=1 to n do fact := fact*i;
        factorielle := fact;
end;
begin
   for i:=1 to 10 do
     writeln ('La factorielle de ', i:2, ' est ', factorielle ( i ) :8:0 );
end.
2. a) procedure remplissage;
var i : integer;
begin
        for i:=1 to 100 do
        begin
```

```
                writeln('Entrez la ', i:3,' ème valeur du tableau');
                readln( tab[i] );
        end;
end;
b) function comptesupn ( n : integer ) : integer;
var i, cpt : integer;
begin
        cpt := 0;
        for i:=1 to 100 do
                if tab[i] > n then cpt := cpt + 1;
        comptesupn := cpt;
end;
c) function somme : integer;
var i, s : integer;
begin
        s := 0;
        for i:=1 to 100 do
                s := s + tab[i];
        somme := s;
end;
d) function moyenne : real;
begin
        moyenne := somme / 100;
end;
e) { Bloc principal }
begin
        remplisage;
        writeln ('Nombre de valeurs supérieures à 10 : ', comptesupn (10) );
        writeln ('Moyenne : ', moyenne :6:2 );
end;
```

3. Solution non fournie. Pourra faire l'objet d'un T.P..

```
4. a) function demandevaleur (inf : integer; sup : integer) : real;
var x : real;
begin
        writeln('Entrez une valeur entre ',inf:2,' et ', sup:2); readln(x);
        while (x < inf) or (x > sup) do
        begin
                writeln('Cette valeur n"est pas bonne, recommencez !'); readln(x);
        end;
        demandevaleur := x;
end;

b) function distance ( x1, y1, x2, y2 : real ) : real;
begin   distance := sqrt ( sqr(x2-x1) + sqr(y2-y1) );
 end;

function estisocele (p1, p2, p3 : tabpoint) : boolean;
var d1, d2, d3 : real;
```

```
begin
        d1 := distance ( p1[1], p1[2], p2[1], p2[2] );
        d2 := distance ( p2[1], p1[2], p3[1], p3[2] );
        d3 := distance ( p1[1], p1[2], p3[1], p3[2] );
        if (d1 = d2) or (d1 = d3) or (d2 = d3)then estisocele := true
                                                    else estisocele := false;
end;
c) function estequilateral (p1, p2, p3 : tabpoint) : boolean;
var d1, d2, d3 : real;
begin
        d1 := distance ( p1[1], p1[2], p2[1], p2[2] );
        d2 := distance ( p2[1], p1[2], p3[1], p3[2] );
        d3 := distance ( p1[1], p1[2], p3[1], p3[2] );
        if (d1 = d2) and (d1 = d3)      then estequilateral := true
                                        else estequilateral := false;
end;
d) procedure affichetriangle ( ppte : string );
var     ligne, colonne, ajout, nbcol : integer;
begin
        writeln;
        if (ppte = 'quelconque') then ajout := 5;
        if (ppte = 'isocèle') then ajout := 2;
        if (ppte <> 'équilatéral')
        then begin      { On traite les 2 premiers cas ensembles }
                nbcol := 1;
                for ligne := 1 to 7 do
                begin
                        for colonne :=1 to nbcol do
                                write ('*');
                        nbcol := nbcol + ajout;
                        writeln;
                end;
             end
        else begin      { Si c'est équilatéral, solution pas plus bête qu'une autre ...}
                writeln('*');
                writeln ('*****');
                writeln ('*********');
                writeln ('***********');
                writeln ('*********');
                writeln ('*****');
                writeln ('*');
             end;
end;
```

8 Tris, recherches

8.1 Exemples de tris

Le tri est un traitement informatique utilisé très souvent, notamment dans les bases de données. Il s'agit de classer des informations d'un même type selon un ordre donné. Il existe de nombreuses méthodes de tri, nous en présentons 2.

Tri par sélection. On recherche le minimum de la liste, on le permute avec l'élément situé à la première place, et on recommence sur la fin de la liste.

```
program triselection;  { Exemple de tri par sélection avec un tableau de 10 entiers. }
var     i , j, memoire, min, positionmin : integer;
        t : array[1..10] of integer;
begin
        { Demande des 10 valeurs à l'utilisateur }
        for i:=1 to 10 do
        begin
                writeln('Entrez la valeur numéro ',i:2); readln( t[i] );
        end;

        { Tri par sélection successive des minimums }
        for i:=1 to 9 do { Lorsque le 9ème minimum est placé, le 10ème, seul, l'est aussi ! }
        begin
                min := t[i];      { Initialisation avec le premier élément non classé }
                positionmin := i;       { On retient la position du minimum }
                for j:=i+1 to 10 do     { On va regarder les autres éléments du tableau }
                        if (t[j] < min)
                        then begin
                                min := t[j];      { Mise à jour du minimum }
                                positionmin := j; { On retient la position de ce minimum }
                             end;
                memoire := t[i];                { On va faire une permutation des valeurs }
                t[i] := min;                    { Placement du minimum en position i }
                t[positionmin] := memoire;    { Replacement de l'ancien t[i] }
        end;

        { Affichage des 10 valeurs après classement }
        for i:=1 to 10 do writeln (t[i]);
end.
```

Tri à bulles. De façon imagée, ce tri consiste à faire passer en première position la valeur minimum, en la permutant avec la valeur voisine de façon itérative, telle une bulle remontant lentement à la surface. On procède ensuite de même avec tout le tableau moins la première valeur pour placer la deuxième plus petite valeur et ainsi de suite.

La suite d'instructions permettant de trier le même tableau t que précédemment est présentée ci-dessous, i, j et memoire étant des variables entières jouant des rôles similaires.

```
{ Classement des 10 valeurs selon la méthode du tri à bulles }
for i:=1 to 9 do
```

```
for j:=9 downto i do
        if (t[j+1] < t[j])
        then begin              { Permutation : on fait passer à gauche le minimum }
            memoire := t[j];
            t[j] := t[j+1];
            t[j+1] := memoire;
          end;
```

8.2 Recherches

Premier cas : tableau non trié.
Pour rechercher une valeur dans un tableau non trié, pour savoir par exemple si cette valeur existe, l'algorithme le plus simple consiste à parcourir le tableau en testant les cases 1 par 1. En supposant qu'un tableau t de n cases est déjà rempli, que la valeur recherchée est x, que i est une variable permettant le parcours du tableau et que vu est un booléen exprimant la présence de x, la recherche de x dans t est rÈalisÈe par les instructions suivantes :

```
vu := false; i:=1;
while (i <= n) and (not vu) do
        if (t[i] = x) then vu := true else i:= i+1;
{ Si vu vaut true, x est présent et si vu vaut false, x est absent }
```

Deuxième cas : tableau trié.
Lorsque vous cherchez un mot dans le dictionnaire, il vous faut peu de temps pour le trouver, car les mots sont triés par ordre alphabétique et votre méthode de recherche est efficace. En informatique, le même problème se pose. Sur un petit nombre de données, le temps de cette recherche est négligeable quelle que soit la méthode, mais sur un grand ensemble de données, il est important de disposer d'une méthode efficace. La méthode dite de dichotomie, présentée ci-dessous, est l'une des plus rapides. Elle consiste à comparer la valeur recherchée x avec la valeur située au milieu du tableau. Si x lui est inférieur, il faut chercher dans la partie gauche du tableau, sinon dans la partie droite. On recommence ensuite la même méthode en considérant une moitié de tableau et en comparant x à la valeur située au milieu, et ainsi de suite. On arrête la recherche lorsque la valeur est trouvée. Cette méthode de recherche dichotomique est illustrée ci-dessous avec un tableau de 100 notes associé à un tableau de 100 noms d'étudiants. Une note donnée nous indique une position dans le tableau, ce qui permet de retrouver le nom de l'étudiant.

```
program dichotomie;
var     notes : array[1..100] of integer;
        noms : array[1..100] of string;
        i, x, notemilieu, milieu : integer;
        fini : boolean;
        borneinf, bornesup : real;
begin
        { Demande des notes et des noms à l'utilisateur }
        writeln('Procédons par ordre, de la plus mauvaise note à la meilleure.');
        for i:=1 to 100 do
        begin
                writeln('Entrez l''étudiant numéro ',i:2); readln( noms[i] );
```

```
            writeln('Entrez la note numéro ',i:2); readln( notes[i] );
        end;

        writeln('Donnez moi une note, je vous retrouve le ou les étudiants.'); readln( x );
        fini := false;    borneinf := 1;  bornesup := 100;
        while (not fini) do
        begin
            milieu := (borneinf + bornesup) div 2;
            notemilieu := notes[milieu];
            if (x = notemilieu) then fini := true
                else if (x < notemilieu) then bornesup := milieu - 1
                                         else borneinf := milieu + 1;
            if (bornesup < borneinf) then fini := true;    { Valeur non trouvée }
        end;

        if (x = notemilieu) then writeln('L"étudiant ', noms[milieu], ' a cette note.');
        i := milieu + 1;                    { Recherches d'éventuelles autres solutions. }
        while (i <= 100) and (x = notes[i]) do[5]
        begin
            writeln('L"étudiant ', noms[i], ' a cette note.');        i := i+1;
        end;
        i := milieu - 1;
        while (i >=1) and (x = notes[i]) do
        begin
            writeln('L"étudiant ', noms[i], ' a cette note.');        i := i-1;
        end;
end.
```

8.3 Exercices

1. Ecrire un tri par sélection pour n données qui place un à un, non pas les minimums, mais les maximums, en conservant l'ordre croissant.

2. Ecrire un tri à bulles pour n données qui fait remonter le maximum en dernière position, puis le deuxième maximum en avant dernière position etc..

3. Lorsqu'on applique le tri à bulles à un ensemble de données déjà un peu trié, il est fréquent de constater que les données sont triées bien avant les dernières itérations. Modifiez l'algorithme du tri à bulles pour minimiser le nombre d'itérations (c'est à dire le nombre de passages dans les boucles), en utilisant l'instruction while.
4. Un tri est d'autant plus efficace qu'il implique un faible nombre de comparaisons entre données. Etudiez le nombre de comparaisons effectuées dans le tri par sélection et le tri à bulles, en fonction du nombre n de données. Quel est l'algorithme le plus efficace ?

5. Soit deux tableaux T1 et T2 de n entiers. Donnez une solution pour placer dans T2 les éléments triés de T1, sans modifier T1 et sans effectuer un tri sur T2. On pourra effectuer une recherche du maximum et le placer dans T2 en position 1, puis du maximum suivant et le

[5]On considère ici que l'évaluation des booléens est optimisée, donc que (x = notes[101]) n'est jamais évalué

placer dans T2 en position 2, et ainsi de suite jusqu'à ce que tous les éléments soient pris. Pour ne pas tenir compte des maxima trouvés précédemment, on pourra utiliser un tableau de booléens, appelé dejavu, qui permettra de mémoriser au fur et à mesure les indices des éléments déjà vus. Par exemple, on initialise dejavu avec des false, puis, si T1[4] est le premier maximum, on affecte true à dejavu[4].
Application : on pourra utiliser des tableaux de 1 à 10 d'entiers, initialisés de façon aléatoire.

6. Exprimez le nombre de comparaisons entre données effectuées dans le pire des cas pour l'algorithme de recherche dichotomique, en fonction du nombre total n de données.

7. Lorsque vous cherchez un mot dans le dictionnaire, votre méthode est sans doute plus rapide qu'une recherche dichotomique. Pourquoi ? En considérant que les mots du dictionnaire pourraient être stockés dans un tableau de string, donnez l'idée générale d'une méthode de recherche qui soit plus efficace que la recherche dichotomique.

8.4 Solutions

1. { Tri de n valeurs par sélection successive des maximums (listing des instructions) }
```
for i:=n downto 2 do
begin
        max := t[1];            { Initialisation avec le premier élément }
        positionmax := 1;       { On retient la position du maximum }
        for j:=2 to i do{ On va regarder les autres éléments du tableau }
                if (t[j] > max)
                then begin
                        max := t[j];     { Mise à jour du maximum }
                        positionmax := j; { On retient la position de ce maximum }
                     end;
        memoire := t[i];                { On va faire une permutation des valeurs }
        t[i] := max;                    { Placement du maximum en position i }
        t[positionmax] := memoire;   { Replacement de l'ancien t[i] }
end;
```

2. { Tri à bulles de n valeurs, les maximums d'abord (listing des instructions) }
```
for i:=10 downto 2 do
        for j:=2 to i do
                if (t[j-1] > t[j]) then begin      { On fait passer à droite le maximum }
                                        memoire := t[j];
                                        t[j] := t[j-1];
                                        t[j-1] := memoire;
                                     end;
```
3. { Tri à bulles de n valeurs, on minimise le nombre d'itérations (listing des instructions) }
```
i := 1;
onpermute := true;                      { onpermute est un booléen }
while (i <=9) and (onpermute) do     { Tant qu'on permute c'est qu'on n'a pas fini }
begin
        onpermute := false;
        for j:=9 downto i do
                if (t[j+1] < t[j])
```

```
                then begin              { Permutation : on fait passer à gauche le minimum }
                    onpermute := true;      { On n'a pas fini, il ne faut pas sortir du while }
                    memoire := t[j];
                    t[j] := t[j+1];
                    t[j+1] := memoire;
                end;
        i := i+1;
end;
```

4. Tri par sélection : nb comparaisons = n (n-1)/2
Tri à bulles : nb comparaisons = n (n-1) / 2
C'est pareil.

5. Solution non fournie. Pourra faire l'objet d'un T.P..

6. Nombre de comparaisons pour la recherche dichotomique dans le pire des cas : $\log_2 (n)$

7. Une méthode simple consiste à mémoriser pour chacune des lettres de l'alphabet quelles sont les bornes inférieures et supérieures des indices des mots commençant par cette lettre. Par exemple, si le mot commence par un c et qu'on sait que les mots commençant par un c sont entre les indices i1 et i2, la recherche est fortement restreinte, ce qui permet de gagner du temps dés le départ. Pour gagner encore plus de temps, on pourrait faire de même en considérant les 2 premières lettres de l'alphabet, voire plus. En ce qui concerne la mémorisation des bornes, on pourrait utiliser deux tableaux de taille 26 d'entiers, en associant à chaque lettre la case dont l'indice correspond à son ordre dans l'alphabet.

9 Les tableaux multidimensionnels

9.1 Les tableaux multidimensionnels

Un tableau peut être de dimension supérieure à 1. La syntaxe générale de déclaration d'un tableau est la suivante :

nom_de_variable : **array** [liste d'intervalles] **of** nom_de_type;

Si la liste d'intervalles comprend plus d'un intervalle, ceux-ci sont séparés par des virgules. Par exemple, tab : array[1..4, 1..3] of integer. Dans ce cas, tab peut être considéré comme une matrice ou une grille de 4 cases par 3. L'accès à une case du tableau se fait simplement en donnant la valeur de chacun des indices. Par exemple tab[4,3] est la dernière case du tableau précédent.

Exemple de programme avec un tableau à 2 dimensions :

```
program equilateral;
var     triplet : array[1..3,1..2] of real;      { Un triplet est constitué de 3 points, chaque }
        nbpoint, coordonnee : integer;           { point ayant 2 coordonnées. }
        dist1, dist2, dist3 : real;
begin
        for nbpoint := 1 to 3 do
        begin
                writeln ('Entrez les coordonnées x, y du point numéro ', nbpoint : 1);
                for coordonnee:=1 to 2 do
                        readln ( triplet [nbpoint, coordonnee] );
        end;
        dist1 := sqrt ( sqr( triplet[1,1] - triplet[2,1]) + sqr (triplet[1,2] - triplet[2,2]) );
        dist2 := sqrt ( sqr( triplet[1,1] - triplet[3,1]) + sqr (triplet[1,2] - triplet[3,2]) );
        dist3 := sqrt ( sqr( triplet[3,1] - triplet[2,1]) + sqr (triplet[3,2] - triplet[2,2]) );
        if (dist1 = dist2) and (dist1 = dist3) then writeln ('Ce triangle est équilatéral')
        else writeln ('Ce triangle n''est pas équilatéral');
end.
```

9.2 Exercices

1. Proposer un type de tableau pour les problèmes suivants :
a) On veut stocker les coordonnées des sommets d'un tétraèdre.
b) On veut stocker le nom des pièces qui sont sur un échiquier (8x8 cases).
c) On veut stocker la table de vérité du "and" du langage Pascal.
d) On veut stocker 35 notes d'étudiants dans les matières math, physique et chimie.
e) Même chose, mais on dispose de 5 groupes de 35 étudiants.
f) On veut stocker les 52 cartes d'un jeu de poker (plusieurs solutions acceptables).

2. Soient 2 tableaux m1 et m2 représentant 2 matrices de 5x5 réels (m1, m2 : array[1..5, 1..5] of real. Donnez la suite d'instructions qui additionne m1 et m2 et place le résultat dans un tableau m3 de même dimension.

3. On considère un tableau d'entiers représentant les notes de n groupes de 30 étudiants. On suppose n déclaré en constante égale à 3. On déclare tabnotes : array[1..30,1..n] of integer;
Ecrire un programme qui :
a) Initialise le tableau de façon aléatoire avec des notes entre 0 et 20.
b) Trouve et affiche la note maximale de chaque groupe.
c) Détermine et affiche la moyenne de chaque groupe ainsi que le numéro (entre 1 et n) du meilleur groupe.

4. Un damier 10 x10 d'un jeu de dames se présente en début de partie de la façon suivante :

10		N		N		N		N		N
9	N		N		N		N		N	
8		N		N		N		N		N
7	N		N		N		N		N	
6										
5										
4		B		B		B		B		B
3	B		B		B		B		B	
2		B		B		B		B		B
1	B		B		B		B		B	
	1	2	3	4	5	6	7	8	9	10

a) Trouver une structure de données adaptée pour ce jeu.
b) Ecrire une procédure init qui place les pions dans le tableau comme indiqué sur la figure.

5. Jeu du démineur.
Un terrain rectangulaire miné est représenté par un tableau de NxM cases d'entiers. Au début du jeu, on placera un nombre X de mines de façon aléatoire dans le terrain : s'il n'y a pas de mine, la valeur de la case doit être égale à -2 et s'il y a une mine, la valeur de la case doit être égale à -1. Ensuite, le joueur doit donner les coordonnées x,y d'une case. S'il tombe sur une case avec -1, il a perdu, sinon, on affecte à la case x,y le nombre de mines présentes dans un carré 3 fois 3 centré en x,y. On affiche alors toutes les cases du terrain avec la convention suivante : si le nombre est négatif (-1 ou -2), on affiche un point d'interrogation, sinon on affiche la valeur de la case. Ensuite, on recommence, le joueur doit donner les coordonnées d'une autre case etc. Le jeu se termine lorsque le joueur tombe sur une mine ou lorsqu'il a trouvé toutes les cases non minées. Le nombre X de mines est demandé à l'utilisateur au début du jeu. N et M pourront être déclarés en constantes, toutes 2 égales à 5 pour commencer.

9.3 Solutions

1. a) tetraedre = array[1..4,1..3] of real; { 4 sommets, 3 coordonnées }
b) echiquier = array[1..8,1..8] of string;
c) tableduand = array[1..2,1..2] of boolean;
d) notes = array[1..35,1..3] of integer; { On suppose que les notes sont des entiers }
e) touteslesnotes = array[1..35,1..5,1..3] of integer; { L'ordre des intervalles importe peu }
f) Ca dépend de ce qu'on veut en faire. Exemple : jeu = array[1..13,1..4] of integer;

2. { Somme de 2 matrices. On utilise 2 entiers nommés i et j }

```
for i:=1 to 5 do
        for j:=1 to 5 do
                m3[i,j] := m1[i,j] + m2[i,j];
```

3.
```
program notation;
const   n = 3;
var     tabnotes : array[1..30,1..n] of integer;
        i,j, max, meilleur : integer;
        moy : array[1..n] of real;
begin
        { a) Initialisation aléatoire }
        randomize;
        for i:=1 to n do
                for j:=1 to 30 do
                        tabnotes [j, i] := round( random(20) );

        { b) Recherche de la note maximale de chaque groupe }
        for i:=1 to n do
        begin
                max := tabnotes [1,i]; { note du 1er étudiant du groupe i }
                for j:=2 to 30 do
                        if (tabnotes[j,i] > max) then max := tabnotes[j,i];
                writeln('Note maximale du groupe ',i:2,' : ', max:2);
        end;

        { c) Calcul de la moyenne de chaque groupe }
        for i:=1 to n do
        begin
                moy[i] := 0;
                for j:=1 to 30 do       { On fait d'abord la somme des notes }
                        moy[i] := moy[i] + tabnotes[j,i];
                moy[i] := moy[i] / 30;
                writeln ('Moyenne du groupe ',i:2, ' : ', moy[i]:5:2);
        end;
        { Recherche du meilleur groupe }
        meilleur := 1;
        for i:=2 to n do
                if (moy[i] > moy[meilleur]) then meilleur := i;
        writeln('Le groupe qui a la meilleure moyenne est le ', meilleur:2);
end.
```

4. Solution non fournie. Pourra faire l'objet d'un T.P.

5. a) array[1..10,1..10] of char;
b) Solution non fournie. Indication : on peut procéder en 3 étapes, les lignes 1 à 4, les lignes 5 à 6 et les lignes 7 à 10, en utilisant le fait que la somme des indices d'une case où il y a un pion est paire.

10 Les procédures et les fonctions, suite

10.1 Passage par valeur ou par référence

Au chapitre 7, nous avons simplifié la déclaration des paramètres d'une procédure. La syntaxe complète est la suivante :

```
procedure nomproc ( liste des paramètres );
var     { déclaration des variables locales }
begin
        { Corps de la procédure : liste d'instructions }
end;
```

La syntaxe de la déclaration de la liste des paramètres est la suivante :
(**var*** nomvar1 : type1 ; **var*** nomvar2 : type2; … **var*** nomvarN : typeN)
* : le var est optionnel

Explications :
Il faut distinguer clairement le paramètre déclaré et l'expression qui lui est associée lors de l'appel de la procédure :
- Si on omet le mot "var" devant le nom du paramètre déclaré, l'expression est évaluée et sa valeur est affectée au paramètre déclaré. Toute modification de la valeur de ce paramètre à l'intérieur de la procédure reste strictement locale. On dit qu'il y a un "passage par valeur".
- Si on met "var" devant le paramètre déclaré, alors il faut que l'expression associée lors de l'appel de la procédure soit une autre variable. A ce moment là, le paramètre déclaré est identifié au paramètre d'appel, même s'il a un nom différent. Ainsi, toute modification de la valeur du paramètre dans la procédure est répercutée sur la variable d'appel. On dit dans ce cas qu'il y a un passage par référence ou par adresse. En effet, c'est parce que le paramètre déclaré et le paramètre d'appel ont la même adresse mémoire (donc la même référence) qu'une modification de l'un entraîne une modification de l'autre.

Exemple :

```
program test; { Juste pour comprendre l'influence de var devant le paramètre déclaré }
var     x, y : integer;

procedure affiche ( var r : integer ; s : integer );
begin
        writeln ( 'x vaut ', x:1, ', r vaut ', r:1, ', y vaut ', y:1, ' et s vaut ', s:1);
        r := 1;  s:=1;
        writeln ( 'x vaut ', x:1, ', r vaut ', r:1, ', y vaut ', y:1, ' et s vaut ', s:1);
end;
begin
        x := 0; y := 0;
        affiche ( x,y );
end.
```

L'affichage à l'écran est : x vaut 0, r vaut 0, y vaut 0 et s vaut 0
x vaut 1, r vaut 1, y vaut 0 et s vaut 1
Commentaires : x et y sont initialisés à 0 au début du programme. A ce moment là, r et s ne sont pas connus. A l'appel de la procédure affiche, x est mis en correspondance avec r, donc

ils valent tous les deux 0 et y est mis en correspondance avec s, donc ils valent 0 également, ce qui explique le premier affichage. Pour r, comme c'est un passage par référence (il y a "var" devant), toute modification est répercutée sur x, donc r et x prennent la valeur 1. En ce qui concerne s, c'est un passage par valeur, donc s prend la valeur 1 mais y n'est pas modifié, ce qui explique le deuxième affichage.

10.2 Exercices

1. On considère le programme suivant :

```
program surprise;
var     x : real;
procedure tcaf ( n : integer; var res : real );
var     i : integer;
begin
        res := 1;
        for i := 1 to n do res := res * i;
end;
begin
        tcaf (10, x);
        writeln ('Le résultat est ', x:8:0);
end.
```

a) Quel est le résultat à l'écran ?
b) Que se passe-t-il si on enlève "var" devant la déclaration du paramètre res ?
c) Que se passe-t-il si on remplace n par x dans toute la procédure tcaf ?
d) Que se passe-t-il si on remplace tcaf (10,x) par tcaf(10,res) ?

2. On considère le programme incomplet suivant :

```
program incomplet;
var     t : array[1..10] of real;
        max, min, moy : real;
begin
        demandevaleurs;
        moy := calculemoyenne;
        trouvemaxetmin ( max, min );
        affichemaxminmoy (max, min, moy );
end.
```

Compléter le programme de façon cohérente en ajoutant les procédures demandevaleurs, trouvemaxetmin et affichemaxminmoy ainsi que la fonction calculemoyenne, dont les noms indiquent clairement le traitement qu'elles effectuent. On limitera les passages par référence autant que possible.

3. Nombres complexes.
a) Ecrire une procédure de nom "conjugué" qui prend comme paramètres les réels a et b définissant un complexe quelconque et qui transforme ce complexe en son conjugué. Ecrire en commentaires pourquoi vous choisissez un passage par valeur ou par référence pour chacun des paramètres a et b. Rappel: le conjugué de a+i*b est a-i*b
b) Ecrire une procédure de nom "produit" qui calcule le produit de 2 nombres complexes. Cette procédure prend comme paramètres les réels a,b,c,d,e,f dont les 4 premiers définissent les 2 complexes a+i*b et c+i*d dont il faut faire le produit, et e et f définissant le complexe

e+i*f résultat de l'opération. Ecrire en commentaires pourquoi vous choisissez un passage par valeur ou par référence pour chacun des paramètres.
c) Ecrire une fonction de nom "module" qui calcule le module d'un nombre complexe. Expliquez en commentaires quels sont les paramètres de cette fonction, ainsi que son type.
Rappel: le module de a+i*b est racine carrée de (a^2+b^2)
d) En s'aidant OBLIGATOIREMENT des procédures et fonctions déjà écrites, compléter la procédure division dont le début est:

```
procedure division (        a,b : real;     (* premier complexe z1 *)
                            c,d : real;     (* deuxième complexe z2 *)
                    var     e,f : real      (* complexe résultat de z1/z2 *)
                   );
(* On utilise le fait que   (a+i*b)/(c+i*d) = ( (a+i*b) * (c-i*d) ) / ( c²+d²)   *)
```

4. Injection, surjection, bijection.
On se donne les déclarations suivantes :

```
const n=10;   m = 10;
type  application = array[1..n] of 1..m;    { 1..m appartient au type intervalle }
      hertz = array[1..m] of integer;
```

Chaque tableau de type application représente une application de domaine {1..n} vers {1..m}. Ainsi, si t est une application, t(x) sera donné par t[x].
a) Ecrire la procédure init (var t : application); qui demande à l'utilisateur les 10 valeurs de l'application.
b) Ecrire la procédure fréquence (t : application; var f : hertz) : boolean; qui, pour tout k de 1..m, place dans f[k] le nombre d'occurrences de k dans t.
c) Ecrire les fonctions suivantes :

```
function injective ( t: application ) : boolean;
function surjective ( t : application ) : boolean;
function bijective ( t : application ) : boolean;
```

Rappels :
Une fonction t est injective si toute image a au plus un antécédent :

$\forall x,y \in \{1.n\}, t(x) = t(y) \Longrightarrow x = y$

Une fonction est surjective si tout élément de l'ensemble d'arrivée est l'image d'au moins un élément de l'ensemble de départ.

$\forall y \in \{1..m\}, \exists x \in \{1..n\}$ tel que $t(x) = y$

Une fonction est bijective si elle est injective et surjective
d) Ecrire le bloc principal en appelant chacune des procédures et fonctions déjà écrites.

10.3 Solutions

1. a) Il s'agit du résultat de factorielle 10. On voit ici que le passage par référence permet de renvoyer un résultat comme le ferait une fonction (voir la fonction de l'exercice 1 du 9.4).
b) Si on enlève "var", la valeur de x n'est pas changée et comme aucune valeur ne lui a été donnée, le résultat est imprévisible.
c) Cela ne change rien, le x de la procédure n'est pas confondu avec le x déclaré en variable globale. Dans le "for i:=1 to x do", c'est donc le x déclaré en paramètre qui est pris en compte.
d) Il y a une erreur de compilation car res n'a pas été déclaré en variable globale et n'est donc pas connu à cet endroit du programme.

2. program incomplet; { Maintenant complet ! }

```
var     t : array[1..10] of real;
        max, min, moy : real;
procedure demandevaleurs;
var     i : integer;
begin
        for i:=1 to 10 do
        begin
                write('Entrez la valeur numéro ', i:2, ' : '); readln ( t[i] );
        end;
end;
function calculemoyenne : real;
var     i : integer;       somme : real;
begin
        somme := t[1];
        for i:=2 to 10 do
                somme := somme + t[i];
        calculemoyenne := somme / 10;
end;
procedure trouvemaxetmin ( var maximum : real; var minimum : real );
var     i : integer;
begin
        maximum := t[1];       minimum := t[1];
        for i:=2 to 10 do
        begin
                if (t[i] > maximum) then maximum := t[i];
                if (t[i] < minimum) then minimum := t[i];
        end;
end;
procedure affichemaxminmoy (max : real; min : real; moyenne : real);
begin
        writeln('Le maximum est ', max:6:2);
        writeln('Le minimum est ', min:6:2);
        writeln('La moyenne est ', moy:6:2);
end;
begin
        demandevaleurs;
        moy := calculemoyenne;
        trouvemaxetmin ( max, min );
        affichemaxminmoy (max, min, moy );
end.
```

3. Solution non fournie.

4. Solution non fournie. Pourra faire l'objet d'un T.P.

11 Type enregistrement et fichiers

11.1 Le type enregistrement (record)

Comment faire pour stocker des données structurées de différents types, par exemple le nom et l'âge d'une centaine de personnes ?
On pourrait utiliser 2 tableaux, le premier contenant des string et l'autre contenant des entiers. Ca n'est pas très pratique. Il existe en Pascal le type record qui permet de grouper au sein d'une même structure des données de n'importe quel type.

Syntaxe générale de la déclaration d'un record :

```
type    nomdelastructure = record
                nomduchamp1 : type1;
                nomduchamp2 : type2;
                ...
        end;
```

Exemples :

```
type    t_personne = record
                nom : string;
                age : integer;
        end;
        t_point = record
                x : integer;
                y : integer;
        end;
```

Ces types peuvent ensuite être utilisés pour définir des variables.
Exemples :

```
var     joueur1, joueur2 : t_personne;
        tabpersonne : array[1..100] of t_personne;
        p1, p2, p3 : t_point;
        quadrilatere : array[1..4] of t_point;
```

Syntaxe générale de l'utilisation d'un record :

Pour donner une valeur à un "champ" du record, on écrit un **.** entre le nom de la variable et le nom du champ :

```
nomdevariable.nomduchamp := expression
```

Pour utiliser la valeur d'un champ dans une expression, c'est pareil, on utilise un **.** entre le nom de la variable et le nom du champ.

Exemples avec les types et variables déclarées ci-dessus.

```
writeln ('Entrez votre nom.');          readln ( joueur1.nom );
writeln('Entrez votre âge.');           readln ( joueur1.age );
tabpersonne[1].nom := 'Dupont';
p3.x := (p1.x + p2.x) div 2;
p3.y := p3.y + 1;
```

11.2 Les fichiers

Les fichiers sont organisés sur le disque, sur la disquette ou sur tout autre support de stockage selon une arborescence logique. Pour désigner un fichier, il faut préciser le chemin complet qui permet l'accès à ce fichier. Par exemple, si vous êtes sur une configuration windows NT classique, c:\winnt\notepad.exe désigne le fichier exécutable appelé "bloc notes".
Il existe 2 façons d'utiliser les fichiers en Pascal, soit en lecture, soit en écriture. Dans le premier cas, on veut uniquement lire les données présentes dans le fichier sans les modifier. Dans le deuxième cas, on veut créer un nouveau fichier ou modifier les anciennes données.
En règle générale, un fichier est considéré comme un empilement de données de même type. Il peut s'agir par exemple d'un fichier d'entiers, de caractères ou d'une structure de type record, mais rarement de plusieurs types à la fois.

11.2.1 Les commandes de base pour manipuler les fichiers

Déclaration d'un fichier en variable :

```
fichier : file of nomdetype;    (* fichier est un nom de variable quelconque *)
```

Procédures prédéfinies pour manipuler les fichiers :

```
assign ( fichier, nomdufichier );    (* nomdufichier est le chemin d'accès (string) *)
reset ( fichier );                   (* Ouverture du fichier en lecture uniquement *)
rewrite ( fichier );                 (* Création du fichier et ouverutre en écriture. *)
write ( fichier, expression );       (* Ecrit la valeur de l'expression dans le fichier
                                        et se positionne sur la donnée suivante. *)
read ( fichier, nomvar );            (* Récupère dans le fichier une donnée du même
                                        type que nomvar et se positionne sur la donnée
                                        suivante .*)
close ( fichier );                   (* Fermeture du fichier, accès terminé. *)
```

fonctions prédéfinies pour obtenir des informations sur les fichiers :

```
filesize ( fichier )        (* Retourne la taille du fichier en octets *)
```

11.2.2 Exemple d'écriture dans un fichier

```
program stockageinfos;
type    t_personne = record
                nom : string;
                age : integer;
        end;
var     f : file of t_personne;
        leclubdes5 : array[1..5] of t_personne;
        i : integer;
begin
        for i:=1 to 5 do
        begin
                write('Nom de la personne n° ', i:2, ' ? '); readln ( leclubdes5[i].nom );
                write ('Quel est son âge ?');   readln ( leclubdes5[i].age );
        end;
```

```
        assign ( f, 'c:\amoi\leclub.txt' );
        rewrite ( f );
        for i:=1 to 5 do
                write ( f, leclubdes5[i] );
        close ( f );
end.
```

11.2.3 Exemple de lecture dans un fichier

```
program recuperationinfos;
type    t_personne = record
                nom : string;
                age : integer;
        end;
var     f : file of t_personne;
        leclubdes5 : array[1..5] of t_personne;
        i : integer;
begin
        assign ( f, 'c:\amoi\leclub.txt' );
        reset ( f );
        for i:=1 to 5 do
                read ( f, leclubdes5[i] );
        close ( f );

        for i:=1 to 5 do
        begin
                write ('Nom de la personne : ', leclubdes5[i].nom );
                writeln ('   Age : ',  leclubdes5[i].age );
        end;
end.
```

11.3 Exercices

1. Les nombres premiers

1.1 Ecrire un programme qui trouve tous les nombres premiers entre 1 et 100 et les affiche à l'écran. Rappel : un nombre est premier si et seulement si il n'admet comme diviseur que 1 et lui-même. Idée de programme Pascal : pour savoir si un entier n est premier, on peut tester tous les entiers k compris entre 2 et n-1. Si le reste de la division entière de n par k est systématiquement différent de 0, alors n est premier.

1.2 Si on veut voir tous les nombres premiers entre 1 et 1000, un seul écran n'est pas suffisant. On va utiliser un fichier. Modifier votre programme pour écrire les nombres premiers entre 1 et 1000 dans un fichier avec extension .pas. Vérifiez votre résultat en chargeant ce fichier grâce à l'éditeur Pascal.

2. On considère la structure de données suivante :

```
const   maxautorise = 25;      (* nombre max théorique d'étudiants dans un TD *)
type    t_matiere = record
                        nom : string;   (* nom de la matière *)
                        note : string;
        end;
        t_etudiant = record
                        nom : string;
                        matieres : array[1..6] of t_matiere;
        end;
        t_groupe = record
                        nbetudiants : integer;  (* nb étudiants du groupe *)
                        array[1..maxautorise] of t_etudiant;
        end;
```

Dans la suite, on suppose que tout a déjà été initialisé, noms d'étudiants, de matières, notes, etc..

2.1 Ecrire une fonction qui prend en paramètre un groupe et renvoie le nombre d'étudiants qui ont une note supérieure ou égal à 10.

function nbmoy (g : t_groupe) : integer;

2.2 Ecrire une procédure qui prend en paramètre un groupe et le nom d'un étudiant et qui affiche toutes ses notes, avec le nom des matières. On suppose que tous les étudiants ont un nom différent. Attention, il se peut que l'étudiant passé en paramètre ne soit pas dans le groupe.

procedure affichenotesetudiant (g : t_groupe; nometu : string);

2.3 Ecrire une function qui calcule la moyenne des notes de tous les étudiants d'un groupe dans une matière donnée.

function moygroupe (g : t_groupe; nommatiere : string) : real;

2.4 Ecrire une procédure qui prend en paramètre un groupe et qui affiche pour chaque étudiant de ce groupe la liste des matières où il n'a pas la moyenne.

procedure affichematierearepasser (g : t_groupe);

12 Exercices de synthèse

12.1 Itérations, tableaux

1. Itérations.
En considérant que chaque lettre correspond à un chiffre entre 0 et 9, on code l'addition suivante :

```
      L U N E
+     M A R S
--------------
  V E N U S
```

Le résultat de l'addition est bien entendu exact et à chaque lettre correspond 1 chiffre unique et réciproquement. Sachant que les valeurs de E et de V peuvent être déduites logiquement et que les valeurs de U, M et S sont respectivement 2, 3 et 9, écrire un programme Pascal permettant d'afficher à l'écran la ou les solutions pour les lettres L, N, A et R. (On peut également arriver au résultat par déduction, mais il y a beaucoup de cas à envisager ... et ce n'est pas la question.)

2. Tableau unidimensionnel et itérations. n personnes, numérotées de 1 à n, sont disposées en cercle. On voudrait éliminer une à une toutes les personnes. Pour cela, on choisit un nombre k et on procède de la façon suivante : on compte de 1 à k les personnes, et la kième, éliminée, quitte le cercle. On continue à compter de 1 à k en recommençant là où on s'était arrêté, et à chaque fois, la kième personne quitte le cercle. Les personnes étant disposées en cercle, on passe automatiquement de la dernière à la première. La dernière personne à rester est déclarée gagnante. Ecrire un programme qui demande à l'utilisateur une valeur pour n et pour k et qui détermine et affiche le numéro du gagnant.

3. Tableaux bidimensionnel. Problème des 8 reines.
On dispose d'un échiquier de 8x8 cases et de 8 pions appelés "reines". On voudrait isoler ces 8 reines sur l'échiquier de sorte qu'il y en ait une par ligne, une par colonne et qu'il n'y ait pas 2 reines sur une même ligne diagonale. En essayant toutes les combinaisons, mais en éliminant rapidement celles qui ne mènent à rien, déterminer une solution à ce problème. On pourra procéder méthodiquement en essayant de placer les reines une par une, en veillant à ce que le nouvel emplacement soit cohérent avec la position des reines déjà placées.

4. Itérations. Un général tente de ranger ses troupes d'une façon originale : il voudrait trouver 2 nombres x et k qui lui permettent d'avoir x colonnes (x>1) avec k soldats dans la 1ère colonne, k+1 soldats dans la 2ème colonne et ainsi de suite jusqu'à la dernière colonne, avec donc toujours 1 soldat de plus dans la colonne suivante. S'il avait 1001 soldats, il aurait pu tout simplement faire 2 colonnes avec 500 soldats dans l'une et 501 soldats dans l'autre. Malheureusement, malgré tous ses efforts, il se rend compte qu'il ne peut réaliser ce rangement avec le nombre n de soldats dont il dispose. Sachant que n est compris entre 1000 et 2000, réaliser un programme qui détermine la solution et trouve ce nombre n.

12.2 Solutions

1.
```
Program addition;
var     U, E, M, S, V : integer;        { Les lettres connues ou }
        L, N, A, R : integer;           { Les lettres à trouver }
```

```
        lune, mars, venus : integer;
begin
        U := 2; M := 3;          S := 9;
        E := 0; { Déduit logiquement de E + S = S }
        V := 1; { Déduit logiquement du fait que la retenue ne peut être que 1 }
        { Essayons tous les cas ! Il reste les chiffres de 4 à 8 }
        for L:=4 to 8 do
          for N := 4 to 8 do
            for A:=4 to 8 do
              for R:=4 to 8 do
                  begin
                          lune := L*1000 + U*100 + N*10 + E;
                          mars := M*1000 + A*100 + R*10 + S;
                          venus := V*10000 + E*1000 + N*100 + U*10 + S;
                          if      (lune + mars = venus)
                            and   (L<>N) and (L<>A) and (L<>R)
                            and   (N<>A) and (N<>R) and (A<>R)
                          then writeln ('Solution : L=',L:1, ', N=',N:1,', A=',A:1,', R=',R:1);
                  end;
end.

2. program ronde;
var     t : array[1..100] of boolean;   { true si la personne est présente, false si on l'élimine }
        position, i, j, k, n : integer;
begin
        writeln('Entrez le nombre de personnes'); readln (n);
        writeln('On avance de combien à chaque fois ?'); readln(k);
        for i:=1 to n do t[i] := true;    { Tous présents ! }
        position := 0;
        for i:=1 to n-1 do       { n-1 itérations pour éliminer n-1 personnes }
        begin
                { On avance de k positions en ne comptant que les présents }
                for j:=1 to k do
                begin
                        position := (position mod n) + 1;
                        while (t[position] = false) do position := (position mod n) + 1;
                end;
                t[position] := false;
                writeln ('On élimine le numéro : ', position :2);
        end;
        { Il n'en reste plus qu'un, il ne reste plus qu'à le trouver ! }
        position := 1;
        while (t[position] = false) do position := position + 1;
        writeln('Celui qui reste est le numéro : ', position:2);
end.
```

3 et 4 : Solutions non fournies.

12.3 Procédures et fonctions

1. On veut effectuer des traitements sur un texte de affiché à l'écran. On a écrit pour cela le programme incomplet suivant :

```
program traitementdetexte;
var     texte : array[0..19, 0..9] of char;      { On dispose de 20 colonnes et 10 lignes }
        x,y : char;
        choix, ligne : integer;
begin
        randomize;
        init;
        affiche;
        choix := -1;
        while (choix <> 0) do
        begin
                writeln('Remplacer un caractère      --> 1');
                writeln('Insérer une ligne blanche   --> 2');
                writeln('Quitter                     --> 0');
                readln(choix);
                if (choix = 1)
                then begin
                        writeln('Quel est le caractère à supprimer ?');      readln(x);
                        writeln('Par quel caractère on le remplace ?');      readln(y);
                        remplacerpar (x,y);
                     end
                else if (choix = 2)
                then begin
                        writeln('A quelle position ?'); readln(ligne);
                        insereligne(ligne);
                     end;
                affiche;
        end;
end.
```

a) Ecrire la procédure "init" qui initialise tout le tableau avec le caractère blanc ' ', puis qui y place 100 lettres de l'alphabet majuscule de façon aléatoire (on rappelle que 'A' = chr(65)).

b) Ecrire la procédure "affiche" qui affiche le texte en respectant l'alignement des lignes et des colonnes.

c) Ecrire la procédure "remplacerpar" avec ses paramètres, qui remplace toute occurrence du caractère désigné par x par le caractère désigné par y.

d) Ecrire la procédure "insereligne" avec son paramètre, qui insère une ligne blanche à la position indiquée et décale tout le reste du texte vers le bas, la dernière ligne du bas étant effacée.

DEUG A1
Septembre 97

Examen d'informatique

(Documents non autorisés)

Exercice 1 : (3 pts) On considère le programme Pascal suivant :

```
program exo1;
var     x,y : integer;
begin
        x := 5;          y:= 0;
        while (x < 10) do
        begin
                x := x+3;       y := y + x;
        end;
        writeln('La valeur recherchée est : ', y );
end.
```

Montrez l'évolution du contenu des variables x,y, puis déduire le résultat à l'écran du programme exo1.

Exercice 2 : (3 pts) Ecrire un programme qui demande à l'utilisateur 4 valeurs réelles correspondant aux coordonnées de 2 vecteurs et qui affiche à l'écran si ces vecteurs sont orthogonaux entre eux ou pas. (2 vecteurs sont orthogonaux si leur produit scalaire est égal à 0)

Exercice 3 : (3 pts) Ecrire une fonction qui calcule le nombre d'entiers multiple de 3 dans un tableau de type tab défini de la façon suivante : tab = array [1..100] of integer;
L'en-tête de la fonction est : function nbmult3 (t : tab) : integer;

Exercice 4 : (3 pts) Ecrire un programme comportant une seule instruction writeln et aucune instruction conditionnelle (if then) et permettant d'afficher à l'écran les 3 lignes suivantes :

```
AAAABBBB
AAABBB
AABB
```

Exercice 5 : (4 pts) Soit t un tableau à 2 dimensions défini en variable globale de la façon suivante :

```
var     t : array[1..10,1..10] of char;
```

Ecrire une procédure qui prend 2 paramètres a et b de type char (caractère) et qui affiche à l'écran les indices de toutes les cases du tableau où a est présent avec b absent des cases du voisinage situées à une distance 1. Les cases situées à une distance 1 sont, lorsqu'elles existent, au-dessus, au-dessous, à droite et à gauche de la case courante.

Exercice 6 : (4 pts) Ecrire un programme qui calcule la valeur approchée à 0,001 près de la solution de l'équation suivante : tan x = - ln x
On pourra utiliser le fait que x est nécessairement dans l'intervalle [0,1; 1] et que - ln x est supérieur à tan x à gauche du point recherché alors que -ln x est inférieur à tan x à droite de ce même point. La valeur de la solution n'est pas demandée (calculatrice inutile !).

DEUG A1, Grenoble 1994

Examen d'informatique

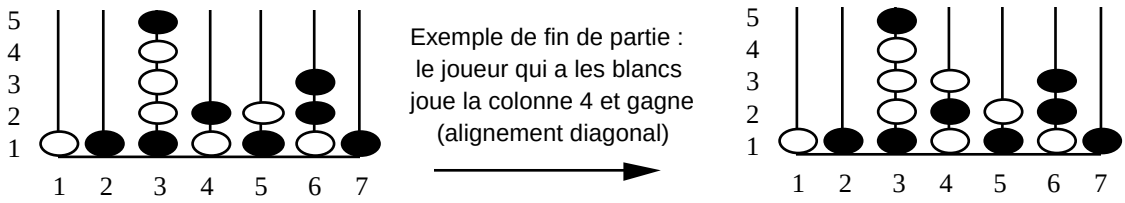
(Durée : 3h)

Cet examen est composé d'un seul problème, mais la plupart des questions sont indépendantes. Vous pouvez donc traiter les questions dans l'ordre que vous voulez, sachant toutefois qu'elles sont présentées dans un ordre croissant de complexité.

But : On désire programmer le jeu Puissance 4 et jouer contre l'ordinateur.

Règles du jeu :

Le jeu puissance 4 est constitué de 7 tiges verticales alignées et de même taille. Dans chaque tige, on peut insérer au maximum 5 boules. Ce jeu se joue à 2. Une personne prend les boules blanches et l'autre les boules noires. Chacun son tour, un joueur insère une boule de sa couleur dans une tige. Toute boule insérée descend le long de la tige jusque en bas ou sur une autre boule placée avant. Le but du jeu est d'aligner 4 boules, soit horizontalement, soit verticalement, soit diagonalement, sans qu'il n'y ait ni vide, ni boule d'une autre couleur intercalée.



Définition des variables globales

- Pour faire jouer l'ordinateur contre un adversaire humain, on va représenter la position des boules par un tableau à deux dimensions d'entiers, placé en variable globale :

  tab : array [1..7, 1..5] of integer;

- Au début toutes les cases de ce tableau seront mises à 0, puis on mettra la valeur 1 en plaçant une boule du joueur et la valeur 2 en plaçant une boule de l'ordinateur.

Par exemple, si au premier coup le joueur insère une boule dans la 3ème tige, il faudra mettre un 1 dans tab[3, 1].

- Pour faciliter les calculs, on va également utiliser un tableau de 7 cases indiquant pour chaque tige quelle est la ligne où se placerait une nouvelle boule :

  lignejouable : array [1..7] of integer;

- Au début du jeu, les 7 cases de ce tableau doivent donc être égales à 1. Puis, si une boule est placée dans une colonne notée x, lignejouable [x] doit être augmenté de 1.
- Pour faire jouer l'ordinateur, on va essayer de noter les 7 coups possibles (1 coup par tige). Pour cela, on va déclarer, toujours en variable globale, un tableau de 7 cases pour stocker la note attribuée à chacun des coups possibles :

  note : array [1..7] of integer;

- Enfin, on va utiliser une variable de type entier pour mémoriser le gagnant. Par convention, ce sera 1 si le joueur gagne, 2 si l'ordinateur gagne et 0 si toutes les cases sont remplies et que personne n'a aligné 4 boules :

  gagnant : integer;

Question 1 :

Construction du bloc principal du programme.
Sachant que :
- "initialisation" est une procédure qui initialise tous les tableaux pour le début du jeu,
- "jeu_adversaire" et "jeu_ordi" sont des procédures permettant le traitement d'un coup pour l'adversaire et pour l'ordinateur,
- "affichage" est une procédure qui affiche l'état du jeu,
- "findepartie" est une fonction booléenne qui rend true si toutes les cases de "tab" sont remplies avec 1 ou 2, c'est à dire si on ne peut plus rajouter de boule,
pour chaque lettre majuscule apparaissant ci-dessous, associez un nom parmi les 4 procédures et la fonction (il peut y avoir plusieurs fois la même).

```
begin
    ……A……;
    gagnant := -1;
    while ( gagnant = -1 ) do
    begin
       (* Jeu de l'adversaire *)
       ……B…… ( gagnant );        (* gagnant est un paramètre de procédure *)
       ……C……  ;
       if ((gagnant <> 1) and (……D…… = false))
       then begin
               (* La partie n'est pas finie, c'est à l'ordinateur de jouer *)
               ……E…… ( gagnant );   (* gagnant est un paramètre de procédure *)
               ……F……;
            end;
       if ((gagnant = -1) and (……G…… = true)) then gagnant := 0;
    end;
    if (gagnant = 1) then writeln ('Vous avez gagné');
    if (gagnant = 2) then writeln ('L'ordinateur a gagné');
    if (gagnant = 0) then writeln ('Egalité');
end.
```

Question 2 :
La procédure "initialisation" place 0 dans toutes les cases du tableau tab et 1 dans toutes les cases du tableau lignejouable. Ecrire la suite d'instructions de cette procédure dont le corps est le suivant :

```
    procedure initialisation;
    var colonne, ligne : integer;
    begin
       ……
    end;
```

Question 3 :
Pour effectuer un affichage simple, on peut se contenter des valeurs 0, 1 et 2 du tableau "tab". Compléter la procédure "affichage" de telle sorte que les valeurs du tableau "tab" apparaissent à l'écran comme les boules du schéma de la première page, avec lignes et colonnes bien ordonnées.

```
    procedure affichage;
    var         ……
```

```
begin
   writeln;
   for ligne := …… downto ……  do
   ……
      for colonne := …… to ……  do
            write ( …… );
   ……
```

Question 4 :

La fonction booléenne "findepartie" retourne true si toutes les cases du tableau sont remplies avec 1 ou 2 et false sinon. Plutôt que de parcourir tout le tableau tab, il est en fait plus astucieux de vérifier qu'il existe au moins une colonne jouable avec l'aide du tableau "lignejouable". Pour cela, il suffit donc de parcourir le tableau "lignejouable" et de vérifier qu'au moins une des cases de ce tableau est inférieure ou égale à 5. Ecrire la fonction findepartie dont l'en-tête est :

```
function findepartie : boolean;   (* aucun paramètre *)
```

Question 5 :

Pour faire jouer l'ordinateur ou pour vérifier le jeu de l'adversaire, il est nécessaire d'avoir des informations sur le nombre de boules alignées dans chaque direction.

Par exemple, il serait intéressant de savoir combien de boules seront alignées verticalement lorsque l'on place une nouvelle boule à une position donnée. Bien entendu, on ne cherche que les alignements sans interruption, il ne faut donc pas qu'il y ait de boule adverse intercalée.

Pour cela, écrire une fonction "alignés_verticaux" qui prend en paramètres un numéro de ligne, un numéro de colonne et le numéro d'un joueur (1 pour l'adversaire et 2 pour l'ordinateur), et qui fournit le nombre de boules de la même couleur, alignées verticalement sans interruption, y compris la boule désignée par la ligne et la colonne. Par exemple, avec le schéma de la première page, si nous considérons que l'ordinateur a les noirs, alignés_verticaux (4, 6, 1) vaut 1 et alignés_verticaux (4, 6, 2) vaut 3. L'en-tête de la fonction est :

```
function alignés_verticaux ( ligne : integer;  colonne : integer;
                                    numérojoueur : integer              ) : integer;
```

Question 6 :

Comme la question 5, mais cette fois-ci, on cherche le nombre de boules alignées horizontalement.

Pour cela, écrire une fonction "alignés_horizontaux" qui prend en paramètres un numéro de ligne, un numéro de colonne et le numéro d'un joueur (1 pour l'adversaire et 2 pour l'ordinateur), et qui fournit le nombre de boules de la même couleur, alignées horizontalement sans interruption, y compris la boule désignée par la ligne et la colonne.

Question 7 :

Comme la question 6, mais cette fois-ci, on cherche le nombre de boules alignées diagonalement. On s'intéresse aux 2 diagonales, il y a donc 2 fonctions à écrire.

Ecrire une fonction "alignés_diagonaux1" qui prend en paramètres un numéro de ligne, un numéro de colonne et le numéro d'un joueur (1 pour l'adversaire et 2 pour l'ordinateur), et qui fournit le nombre de boules de la même couleur, alignées sur 1 diagonale sans interruption, y compris la boule désignée par la ligne et la colonne.

Ecrire également la fonction "alignés_diagonaux2" pour l'autre diagonale.

Les fonctions des questions 6 et 7 ayant beaucoup de ressemblances, on pourra donner uniquement les modifications pour passer d'une fonction à une autre, en précisant clairement ce qui est conservé et ce qui ne l'est pas.

Question 8

Ecrire la procédure "jeu_adversaire". Le ou les paramètres de cette procédure peuvent être déduits de la définition du bloc principal, présenté à la question 1. Il s'agit ici de demander à l'utilisateur dans quelle colonne il veut jouer, de mettre à jour "tab" et "lignejouable", puis de tester si le joueur a gagné pour mettre à jour une certaine variable. On rappelle que pour gagner, il faut aligner 4 boules dans 1 des 4 directions. Il est donc conseillé d'utiliser les fonctions précédentes.

Question 9

On suppose qu'il existe une procédure "évaluation_des_coups" qui met à jour le tableau "notes" défini en variable globale. On va construire ici la procédure "jeu_ordi", qui va donc d'abord appeler la procédure "évaluation_des_coups", puis qui va parcourir le tableau notes et qui va retenir la meilleure note et la colonne qui y correspond. C'est dans cette colonne que l'ordinateur va jouer, il faut donc mettre à jour "tab" et "lignejouable". Enfin, on considère que la note 100 indique un coup gagnant permettant d'aligner 4 boules. Si la meilleure note vaut 100, il ne faut donc pas oublier une certaine mise à jour. L'en-tête de "jeu_ordi" peut être déduit de la définition du bloc principal.

Question 10

Il reste à écrire la procédure "évaluation_des_coups". Nous proposons une stratégie relativement simple pour attribuer une note à chaque colonne :
Pour chaque colonne, si celle-ci est pleine, on donne la note -1, sinon si le coup joué permet un alignement de 4 boules, on donne la note maximale de 100, sinon, si ce coup empêche l'adversaire d'effectuer au coup suivant un alignement de 4 boules, on donne la note 50, sinon, on donne la note N définie comme suit :

N = (Somme des alignements de l'ordinateur passant par cette position)
+ (somme des alignements de l'adversaire s'il jouait cette position)
- (valeur absolue de l'écart entre la colonne jouée et la colonne 4).

Il faut donc écrire cette procédure en s'aidant des fonctions déjà écrites. A titre d'information, cette stratégie se révèle déjà assez efficace et il faut réfléchir un peu pour battre l'ordinateur. Il est cependant facile d'améliorer cette stratégie à l'aide de quelques tests supplémentaires qui ferait alors de l'ordinateur un adversaire redoutable.

12.4 Solutions

```
1. a) procedure init;
var     i,j, ligne, colonne : integer;
begin
        for i:=0 to 19 do
                for j:=0 to 9 do
                        texte[i,j] := ' ';


        i := 0;
        while (i < 100) do
```

```
        begin
                colonne := round ( random(19) );
                ligne := round( random(9) );
                if (texte[colonne, ligne] = ' ')
                then begin
                        i := i + 1;
                        texte[colonne, ligne] := chr ( 65 + round(random(25)) );
                     end;
         end;
end;

b) procedure affiche;
var     ligne, colonne : integer;
begin
        writeln;
        for ligne := 0 to 9 do
        begin
                for colonne := 0 to 19 do write ( texte[colonne, ligne] );
                writeln;
        end;
end;

c) procedure remplacerpar ( x : char; y : char );
var     ligne, colonne : integer;
begin
        for ligne := 0 to 9 do
          for colonne := 0 to 19 do
                if (texte[colonne, ligne] = x) then texte[colonne,ligne] := y;
end;

d) procedure insereligne ( l : integer );
var     ligne, colonne : integer;
begin
        { Commençons par décaler vers le bas }
        for ligne := 9 downto l+1 do  { Attention, dans l'autre sens, ça ne marche pas ! }
          for colonne := 0 to 19 do
                texte[colonne, ligne] := texte[colonne, ligne-1];
        { Insérons la ligne blanche à la ligne l }
        for colonne := 0 to 19 do texte[colonne, l] := ' ';
end;
```

2 et 3 : Solutions non fournies.

Conclusion

Il reste sans aucun doute encore beaucoup de choses à apprendre, que ce soit en algorithmique et programmation, ou en informatique en général. Certaines notions comme le système d'exploitation, le microprocesseur, Internet et l'intelligence artificielle conservent inévitablement un certain flou. Et encore avons-nous passé sous silence de nombreuses expressions du jargon informatique, comme une mémoire cache, une rapidité de calcul exprimée en gigaflops, une architecture RISC ou un contrôleur SCSI. Seule une lecture assidue de livres et de revues spécialisées permet de se familiariser petit à petit avec l'univers informatique.

En deuxième année, nous verrons comment récupérer des données stockées dans un fichier sur le disque dur, comment faire des fonctions ou procédures récursives, ce qu'est un automate, un graphe et plein d'autres choses intéressantes. En deux ans, on apprend finalement bien peu de choses, et cela ne satisfera pas le curieux qui ne cesse de s'interroger sur le fonctionnement des ordinateurs. Si vous êtes de ceux qui cherchent réponse à tout, ou seulement à une question bien précise, alors n'hésitez pas, allez voir l'enseignant et posez lui la question, il se fera une joie de vous répondre… s'il connaît la réponse !