Programmation et langages
Lazarus (Free Pascal)

5e année Transition - Informatique

Pierrard - Virton

Eric Henrotte

TABLE DES MATIÈRES

TABLE DES FIGURES

INDEX

Première partie

Notions essentielles

CHAPITRE 1

INTRODUCTION

1.1 Définitions

1.1.1 Ordinateur

Dans le cadre de l'utilisation que l'on va en faire, un **ordinateur**, c'est une machine à *traiter des informations*, de manière *formelle*, pour autant qu'on lui ait indiqué, dans le détail, *comment mener à bien* ce traitement.

1.1.2 Traitement formel d'information

L'ordinateur va donc s'atteler au traitement d'informations par exemple :

- additionner deux nombres ;
- calculer une moyenne ;
- chercher le plus grand d'une série de nombres ;
- trier un paquet de cartes numérotées qui se présentent dans le désordre ;
- transcrire en toutes lettres un nombre écrit en chiffres ;
- chercher le numéro de téléphone d'un abonné ;
- insérer un nouvel élève ;
- conjuguer un verbe régulier du premier groupe à tous les temps de l'indicatif ;
- lire un texte et fournir la fréquence des divers mots le composant ;
- centrer un titre lors de la dactylographie d'un document ;
- ...

Le premier ensemble de tâches concerne des traitements de nombres et les manipulations de données numériques sont, par essence, formelles. Le processus d'addition de deux nombres n'a que faire du "sens" qui serait attaché aux quantités à additionner, qu'il s'agisse de poids, de sommes d'argent ou d'âges.

Le deuxième exemple de tâches correspond lui aussi, même si les informations ne sont plus numériques, à des traitements formels : trier, chercher un numéro de téléphone, classer, ... sont des actions qui ne font appel qu'à la forme des données manipulées. ANTOINE

précèdera toujours BENOIT dans un classement alphabétique et c'est l'apparence externe de ces prénoms qui permettra de dire que « AAFFRR » n'est pas un prénom. L'ordinateur, ne travaillant qu'avec la forme des données qui lui sont transmises, classera « AAFFRR » avant ANTOINE sans se soucier du sens du mot « AAFFRR » dans la liste de prénoms.

Le troisième exemple de tâches concerne la manipulation formelle de texte. Le texte est considérer comme une chaîne (ou succession) de caractères et non de mots. En aucun cas l'ordinateur ne se souciera du sens d'un texte. Pour s'en rendre compte, il suffit de vérifier les propositions qu'un correcteur orthographique peut proposer pour un mot erroné. Par exemple, si j'écris « je fais une phôte d'orthographe », le mot « phôte » est repéré dans le texte comme étant incorrect, et les propositions de corrections sont « hôte » ou « hôtes » ! Par contre, rien dans la phrase « il faut pou voir se rencontrer » n'éveille une réaction de l'ordinateur ; les mots « pou » et « voir » se trouvant dans la liste des mots acceptables.

1.1.3 Opposition « utilisateur – programmeur »

Schématiquement, on peut représenter les choses de la manière suivantes :

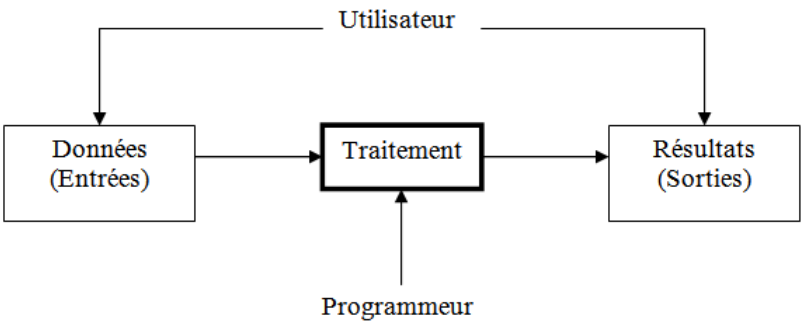


FIGURE 1.1 – La programmation

L'**utilisateur** voit dans l'ordinateur, équipé des programmes convenables qui le feront agir, un outil pour l'aider à accomplir sa tâche (présenter un document, calculer des trajectoires balistiques, gérer les opérations bancaires, dessiner, ...). Ce qui l'intéresse, c'est essentiellement : « quelles données dois-je fournir ? » et « quels résultats suis-je en droit d'attendre après le traitement qui y sera appliqué ? ». Pour le reste, l'ordinateur équipé du programme (logiciel) qui le gouverne pendant l'utilisation, c'est une boîte magique.

Le **programmeur** (ou plutôt l'analyste-programmeur) s'intéresse d'abord au traitement proprement dit. Pour lui, l'ordinateur est d'abord un exécutant à qui il va falloir fournir une marche à suivre (programme) expliquant le traitement à effectuer.

1.1.4 Algorithmes et programmation

Démarche type de résolution d'un problème en informatique

La **programmation** implique une démarche de résolution de problème qui permet d'appréhender la complexité. Cette démarche, qui doit être mise en œuvre pour tous les problèmes de programmation et qui doit être accompagnée d'un rapport final sur le travail effectué, peut être décomposée de la manière suivante :

- discussion, élaboration et rédaction du cahier de charge ;
- analyse du problème ;
- rédaction de l'algorithme (langage de description des algorithmes) ;
- transcription de l'algorithme dans un langage (lisibilité : commentaires, indentation) ;
- tests et simulations ;
- corrections, validation, débogages.

Programmer c'est faire faire...

Il s'agira à chaque fois pour le programmeur, non d'exécuter des tâches lui-même, mais de **les faire exécuter**. C'est cela programmer !

> Il s'agit, non seulement d'être capable de mener à bien **soi-même** une tâche donnée, mais d'expliquer à **un autre** (l'ordinateur) comment il doit s'y prendre pour exécuter cette tâche à sa place.

Le problème principal du programmeur est de parvenir à décrire correctement et le plus simplement possible la suite des actions élémentaires permettant d'obtenir, à partir des données fournies, les résultats escomptés. Un **algorithme** est une suite d'opérations permettant de résoudre un problème spécifique.

Deux représentations des algorithmes

1. Pseudo code

Lire N
$S \leftarrow 0$
$I \leftarrow 1$
tant que $I \leq N$ **faire**
  $S \leftarrow S + I$
  $I \leftarrow I + 1$
fin
Afficher S

FIGURE 1.2 – Pseudo code

2. Organigramme

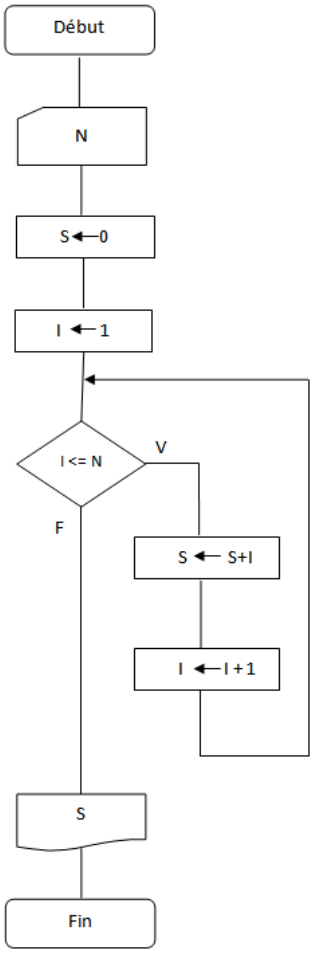


FIGURE 1.3 – Organigramme

1.2 Installation de Lazarus

Il suffit de se rendre sur le site *http ://www.lazarus-ide.org/*, de télécharger puis d'exécuter.

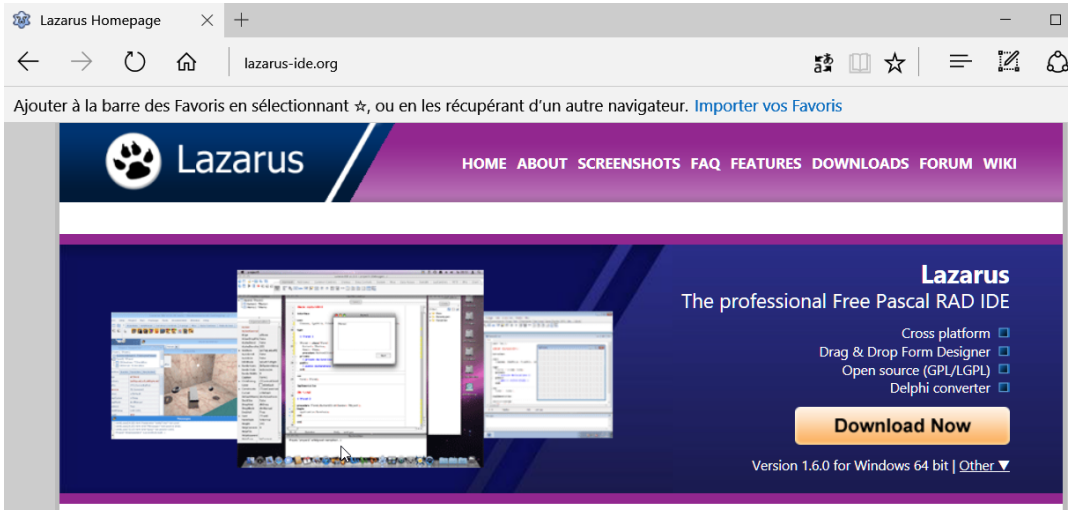


FIGURE 1.4 – Homepage Lazarus

Lazarus sera utilisé pour écrire des programmes en mode console (aucun "widget" comme des boutons, cases à cocher, ...).

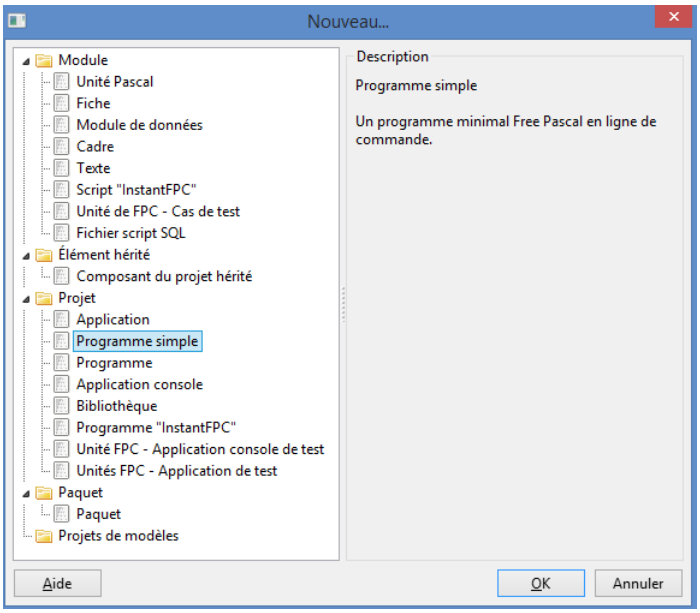


FIGURE 1.5 – Projet Lazarus : programme simple

On obtient la fenêtre suivante où il est possible d'écrire du code

FIGURE 1.6 – Fenêtre du code

qu'on exécutera avec 

CHAPITRE 2

CONCEPTS DE BASE

2.1 Les identificateurs

Chaque fois que l'on fait référence à un objet du programme (une variable, une constante, le nom d'une procédure...), c'est par l'intermédiaire d'un nom, appelé *identificateur*. Un identificateur est une suite de caractères de longueur non limitée dont le premier doit obligatoirement être une lettre (non accentuée) ou le caractère de soulignement (_). Après cette première lettre peuvent figurer exclusivement des chiffres, des lettres (non accentuées) ou des caractères de soulignement (dans un ordre quelconque). Le caractère de soulignement est souvent utilisé dans le but d'améliorer la lisibilité. Voici quelques exemples d'identificateurs :

```
A       Epsilon45     revenu_brut
```

Le choix des identificateurs peut également favoriser la lecture et la compréhension d'un programme. II est déconseillé d'utiliser des identificateurs trop courts, ne suggérant aucune signification (par exemple A, X2, z), ainsi que des identificateurs trop longs, nécessitant plus de travail lors de l'écriture, et surtout n'offrant pas forcément une meilleure lisibilité.

Le langage Pascal ne distingue pas les minuscules des majuscules qui forment un identificateur. Ainsi les trois noms qui suivent désignent le même objet :

```
Rendement     rendement     RENDEMENT
```

2.2 Constantes et variables

Le langage Pascal fait une distinction entre constantes et variables. On utilise une constante chaque fois qu'un objet garde la même valeur tout au long d'un programme. Une constante reçoit une valeur au moment de la compilation du programme et cette valeur ne peut pas être modifiée. Une variable sera au contraire utilisée comme un objet dont la valeur peut être modifiée durant l'exécution du programme.

2.2.1 Définition

Une variable est une association entre une place de la mémoire et un nom. Une variable sert à stocker des données, des résultats, des informations que le programme pourra réutiliser ensuite. Une variable sert à stocker une valeur. Cette valeur peut être (type) :

- un nombre : 12 ou 75.5
- une chaine de caractères : 'Arts & Métiers – PIERRARD'
- ...

Remarques :

- la mémoire (mémoire centrale ou RAM) utilisée durant l'exécution d'un programme peut être comparée à un bureau avec de nombreux tiroirs, espaces de travail, ...
- chaque case de la mémoire est repérée par un numéro distinct : son adresse ;
- cette mémoire est différente du disque dur qui permet de retrouver ses fichiers après avoir éteint et rallumé l'ordinateur [1].

Une variable s'utilise souvent en 3 étapes :

- déclaration : définir le type de la variable.
- initiation : donner une valeur à la variable.
- utilisation (de la valeur de la variable) : par exemple pour l'afficher, pour calculer, ...

2.2.2 Déclaration

Comme tous les objets définis par le programmeur, les constantes et les variables doivent être déclarées avant leur utilisation. La déclaration d'une variable comporte deux parties : le **nom de la variable** [2] et son **type**. Voici un exemple de déclaration de constantes et de variables :

```
procedure premier;
const nul = 0;
   code = 'secret';
   zede = 'z';
var age  : integer;
   salaire : real;
   sexe  : char;
begin
   // on utilise les variables et constantes ici
end.
```

Dans le cas des constantes, l'identificateur est suivi du signe "=" et de la valeur que l'on associe à la constante. Pour les variables, l'identificateur est suivi du signe " :" et du type de la variable. Chaque déclaration de constante ou de variable se termine par un point virgule. Si plusieurs variables sont du même type, il est possible de les déclarer sous forme de liste d'identificateurs (séparés par une virgule) suivie du signe " :" et du type des variables :

```
var   age, enfants, voitures : integer;
   salaire, taille : real;
```

1. La mémoire centrale (RAM) est dite **volatile** tandis que le disque dur est une mémoire **permanente**
2. Rappel : le nom d'une variable doit commencer par une lettre ou _

Les déclarations précédentes peuvent également s'écrire :

```
var   age    : integer;      { du père }
   enfants : integer;    { à charge    }
   voitures : integer;   { disponibles }
   salaire : real;       { maximum     }
   taille : real;     { du capitaine }
```

2.3 Commentaires

Dans le but d'améliorer la lisibilité et la compréhension des programmes, il est **fortement conseillé** d'introduire des commentaires. Un commentaire est un texte explicatif plus ou moins long, placé dans le programme, et ignoré par le compilateur. Les commentaires sont donc totalement invisibles et inutiles dans la phase de compilation et d'exécution d'un programme, mais d'une importance primordiale dans les phases de conception, de mise au point et de maintenance.

Ces explications, visibles uniquement dans les fichiers source (contenant le texte du programme), sont essentiellement destinées aux personnes susceptibles d'analyser un programme ou de lui apporter des modifications. Dans la plupart des cas, les commentaires sont destinés à l'auteur du programme. Mais dans tous les cas où un programme est réalisé en équipe, ou repris par d'autres personnes que l'auteur, les commentaires doivent être une aide à la compréhension, surtout dans certains passages peu évidents.

II n'est pas rare de rencontrer des programmes où les commentaires occupent plus de place que les instructions elles-mêmes. II ne faut cependant pas oublier de mettre à jour les commentaires lors de la modification de tout ou partie du programme. Un commentaire devenu caduque ou qui n'a pas été mis à jour perd son utilité et peut même devenir un obstacle à la compréhension d'un programme.

En Pascal, les commentaires sont reconnus comme tels par le compilateur grâce à des marques de début et de fin qui sont soit des accolades { }, soit les symboles (* et *). II est possible d'imbriquer un type de commentaire dans l'autre type de commentaire. En voici quelques exemples :

```
(*
   Bonjour.
   Ce programme affiche un message de bienvenue.
*)
(* Programme écrit par : Durant Eva  le 25/09/2012 *)
{ imbrication de commentaires (* très drôles *)}
```

Il existe un troisième type de commentaire issu du langage C : chaque ligne commençant par // est considérée comme un commentaire. Dans ce cas, le commentaire ne peut s'étendre que sur une ligne et ne possède qu'une marque de début :

```
Writeln('Bonjour'); // Affiche "Bonjour" à l'écran
```

2.4 Affectation

L'affectation (ou assignation) est l'une des instructions les plus importantes en Pascal. Elle permet de placer une valeur, qui est le résultat de l'évaluation d'une expression, dans une position mémoire référencée par une variable :

```
variable := expression ;
```

où variable est l'identificateur d'une variable qui a été déclarée auparavant.
L'expression peut être une simple variable ou constante, ou bien une combinaison de constantes, variables et opérateurs arithmétiques. Le symbole

$:=$

indique l'affectation et pourrait être traduit par "**prend la valeur de**". Les deux signes qui le composent doivent être accolés.

En Pascal, on a voulu éviter toute confusion entre le symbole de l'affectation (" :=") et celui de l'égalité (représenté par le signe "="). Lors d'une affectation, l'expression qui se trouve à droite du symbole := est évaluée, ensuite seulement le résultat de l'évaluation est affecté à la variable située à gauche du symbole :=. De plus, il est impératif que le type de l'expression soit le même que le type de la variable à laquelle on affecte le résultat de l'expression [3].

2.5 Instructions et blocs d'instructions

Les instructions en Pascal sont séparées par des points virgules. Lorsque plusieurs instructions forment logiquement un tout, on est souvent amené à les grouper. On obtient alors un bloc d'instructions. Le début d'un tel bloc est indiqué par le mot réservé *begin*, alors que sa fin est indiquée par le mot réservé *end*. On parle parfois d"'instruction composée" au lieu de "bloc d'instructions". Ceci exprime bien le fait qu'un groupement de plusieurs instructions peut être vu comme une seule instruction (indivisible). Le corps d'un programme Pascal est lui-même un bloc d'instructions. Voici un exemple de bloc d'instructions :

```
begin
   age    := 55;
   no := age * 10;
end;
```

Lorsque l'on écrit ses premiers programmes, les points virgules posent parfois des problèmes. La règle concernant les points virgules est la suivante :

> Chaque instruction doit se terminer par un point-virgule.

Toutefois, le point-virgule peut être omis s'il est suivi des mots réservés *end* ou *until*. Il **doit être omis** s'il est suivi du mot réservé *else* (sauf s'il s'agit d'une structure sélective *case...of*).

3. Il y a parfois des conversions implicites de types, par exemple, un entier en réel.

En suivant cette règle, l'exemple précédent aurait pu s'écrire de la manière suivante

```
begin
    age := 55;
    no := age * 10
end;
```

Bien que cette forme d'écriture soit tout à fait correcte, il est conseillé de l'éviter au profit de la première citée. L'économie de points virgules peut parfois conduire à des erreurs de syntaxe lors de la modification d'un programme. Ajoutons, par exemple, une ligne à la fin du bloc d'instructions (avant le end). Le risque d'oublier de placer un point virgule à la fin de la ligne qui précède donne le résultat suivant :

```
begin
    age := 55;
    no := age * 10 // il manque un ; ici
    no := no - age // ligne ajoutée
end;
```

Ce fragment de programme n'est pas correct, car il manque le séparateur entre les deux dernières instructions.

Séquence : l'**ordre** dans lequel les différentes opérations seront écrites indique l'ordre dans lequel elles seront exécutées : de haut en bas et de gauche à droite. Il s'agit d'une **exécution séquentielle** (séquence).

2.6 Types et expressions

Parmi les avantages qu'offre le langage Pascal, on trouve une grande richesse de types de données. Nous avons vu précédemment qu'à chaque variable correspond un type. La notion de type est très importante puisqu'elle détermine la nature et l'ensemble des valeurs que peut prendre une variable. Dans cette section nous nous intéressons aux *types scalaires*. Parmi les types scalaires on trouve les types entiers, réels, booléen et caractère, ainsi que les types énumérés ou définis par l'utilisateur. Nous étudierons également les expressions qu'il est possible de construire sur la base de chacun de ces types.

Une expression est une combinaison d'objets qui sont des opérateurs, des opérandes et des parenthèses. Nous serons amenés à évoquer la représentation interne des nombres, ou du moins la place mémoire occupée par une variable d'un type donné. En informatique, l'unité de mesure de la capacité mémoire est l'octet (byte) ; un octet étant lui-même composé de 8 chiffres binaires (bits) pouvant prendre chacun la valeur 0 ou 1.

2.6.1 Les types entiers

II existe plusieurs type permettant de stocker des valeurs entière. Voici leurs caractéristiques :

Type	Intervalle	Format/taille
Byte	$0 .. 255$	1 octet
Shortint	$-128 .. 127$	1 octet
Smallint	$-32768 .. 32767$	2 octets
Word	$0 .. 65535$	2 octets
Longint	$-2147483648 .. 2147483647$	4 octets
Cardinal	Longword	4 octets
Longword	$0 .. 4294967295$	4 octets
Int64	$-2^{63} .. 2^{63} - 1$ $-9223372036854775808 .. 9223372036854775807$	8 octets
QWord	$0 .. 2^{64} - 1$ $0..18446744073709551615$	8 octets
Integer	Longint ($-2147483648 .. 2147483647$)	4 octets

FIGURE 2.1 – Les types entiers

Malgré leurs particularités, les différents types entiers se comportent de la même manière. Pour garder une cohérence dans les types de données, une opération entre deux nombres entiers doit fournir un résultat de type entier. Ceci est effectivement le cas pour :

- l'addition 3 + 5 donne 8
- la soustraction 3 – 5 donne -2
- la multiplication 3 * 5 donne 15

Mais la division (/) pose un problème :

- le quotient de 6 par 4 donne 1.5 qui n'est pas un nombre entier.
- le quotient de 6 par 3 donne 2.0 qui n'est pas un nombre entier.

/ donne toujours un résultat réel (jamais entier)

II a donc fallu implémenter une division, spécifique aux nombres entiers, qui conserve le type des opérandes

La division entière s'écrit **div**

```
6 div 4  donne 1
10 div 3 donne 3
```

Le résultat de cette division correspond à la troncature de la partie décimale. Parallèlement à cette division entière, il est souvent utile de connaître le reste de la division entre deux nombres entiers.

On obtient le reste d'une division entière avec **mod**

```
10 mod 3 donne 1
23 mod 4 donne 3
```

Lorsque des opérateurs et des opérandes sont combinés de manière à former une expression, il est nécessaire d'avoir une convention permettant de l'évaluer. Ainsi l'expression $4 * 2 + 3$ donnera-t-elle le résultat 11 ou bien 20 ? Tout dépend de l'ordre dans lequel sont effectuées les opérations. Les conventions d'évaluation en vigueur en Pascal correspondent à des règles de priorité (cf. Figure 2.5, page 29), semblables à celles qui existent en mathématique :

- les opérateurs div, mod, et * sont prioritaires par rapport aux opérateurs + et - ;
- dans chacune de ces deux catégories les opérateurs ont la même priorité ;
- en cas d'égalité de priorité, les opérations concernées sont effectuées de gauche à droite.

Ainsi :

```
4 * 2 + 3    donne 11
8 + 4 * 3 div 2    donne 14
6 mod 4 * 2 div 3 donne 1
```

Pour modifier ces règles de priorité, il est toujours possible d'utiliser les **parenthèses** :

```
4 * (2 + 3) donne 20
7 div ((5 mod 3) mod 4) donne 3
```

II est également possible de faire précéder un nombre par l'opérateur unaire "**-**" :

```
-4 *12    donne -48
4 *(-5)   donne -20
```

Base 16 on peut écrire un entier en hexadécimal en utilisant le symbole $. Par exemple $FF (255).

Infini on obtient le plus grand entier (*integer*) en écrivant **MaxInt**

2.6.2 Les types réels

Les différents types réels se différencient par leur domaine de définition, le nombre de chiffres significatifs (précision) et la place mémoire occupée. Le tableau suivant résume ces différentes caractéristiques :

Type	Intervalle	Chiffres	Taille en octets
Single	1.5×10^{-45} .. 3.4×10^{38}	7-8	4
Real	5.0×10^{-324} .. 1.7×10^{308}	15-16	8
Double	5.0×10^{-324} .. 1.7×10^{308}	15-16	8
Extended	1.9×10^{-4932} .. 1.1×10^{4932}	19-20	10
Currency	-922337203685477.5808 .. 922337203685477.5807	19-20	8

FIGURE 2.2 – Les types réels

2.6.3 Conversion réels $\Longleftrightarrow$ entiers

1. II existe deux manières de convertir une valeur réelle en valeur entière : l'arrondi et la troncature. Le Pascal dispose de deux fonctions standard qui effectuent ces opérations sur les nombres réels.

La troncature

La fonction *Trunc* agit de manière à tronquer la partie décimale d'un nombre réel. Son utilisation est illustrée par les exemples qui suivent :

```
trunc (4.2) donne 4
trunc (4.99)   donne 4
trunc (-12.6)  donne -12
trunc (3.01)   donne 3

r := 5.67;      { r est de type réel }
i := trunc (r);   { i est de type entier et contient 5}
```

L'arrondi

La fonction *Round* permet d'arrondir un nombre réel à l'entier le plus proche :

```
round (2.99)   donne 3
round (2.5) donne 2
round(2.50000001) donne 3
round (2.499)  donne 2
round (-7.8)   donne -8

r := 4.77;      { r est de type réel }
i := round (r);   { i est de type entier et contient 5}
```

Remarques : partie décimale et fractionnaire

- La fonction *Frac* renvoie la partie décimale d'une expression de type réel. Le résultat est un réel.

  Frac(*pi*) donne 0.14159265358979324

- La fonction *Int* renvoie la partie entière d'une expression de type réel. Le résultat est un **réel**.

  Int(*pi*) donne 3.0

2. La conversion d'une valeur entière en une valeur réelle s'effectue sans passer par des fonctions spécifiques. Lors de l'affectation d'une expression de type entier à une variable de type réel, la conversion de type est implicite. Dans les instructions qui suivent, a et b sont des variables de type entier et c de type réel :

```
a := 6;
b := 2;
c := 2 * a + b;    { c contiendra la valeur 14.0}
```

   Très souvent le programmeur est confronté à des expressions mixtes, contenant aussi bien des valeurs entières que réelles. Dans ce cas, le résultat sera de type réel.

```
c := 15 div a;      // c contiendra 2.0
c := 15 / a;        // c contiendra 2.5
c := 1.5 * (a + b) ; // c contiendra 12.0
```

Infini on obtient le plus grand réel (*real*) en écrivant **Infinity** qui est défini dans l'unité *math* ("uses math" nécessaire au début du programme).

2.6.4 Type booléen (boolean)

L'ensemble des valeurs constituant le type booléen (boolean) est réduit à deux identificateurs de constantes prédéfinies : *true* (vrai) et *false* (faux). On parle souvent de variables ou d'expressions logiques, car le langage Pascal est fortement inspiré de la logique mathématique en ce qui concerne les opérations associées au type booléen. Les constantes et variables de ce type se déclarent de la manière suivante :

```
const demo = true;
var majuscules, termine : boolean;
```

La constante *demo* a la valeur true ; les variables *majuscules* et *termine* peuvent recevoir les valeurs true ou false, selon leur utilisation dans le programme.

Parmi les opérateurs booléens on trouve, dans une première catégorie, les opérateurs relationnels et dans une seconde catégorie, les opérateurs logiques.

- La première catégorie d'opérateurs booléens est constituée des opérateurs **relationnels** :

<	plus petit
>	plus grand
=	égal
<>	différent
<=	inférieur ou égal
>=	supérieur ou égal

FIGURE 2.3 – Opérateurs relationnels

  Ces opérateurs se rencontrent dans les expressions booléennes correspondant à des conditions. Comme nous le verrons plus loin, les conditions interviennent notamment dans les instructions sélectives et répétitives.

- La seconde catégorie d'opérateurs booléens est constituée par des opérateurs purement **logiques**. Ils sont au nombre de quatre, et sont illustrés ci-dessous par des exemples :

not (non) négation logique	si x > 12 est vrai, alors not (x > 12) est faux
and (et) conjonction logique	(x > 2) and (x < 10) est vrai, si (x > 2) est vrai et (x < 10) est vrai
or (ou) disjonction logique	(x < 2) or (x > 10) est vrai, si (x < 2) est vrai ou (x > 10) est vrai
xor (ou exclusif) exclusion logique	a xor b est vrai si a et b n'ont pas la même valeur logique

FIGURE 2.4 – Opérateurs logiques

Comme il est possible, et même très courant, de construire des expressions logiques contenant aussi bien des opérateurs arithmétiques que logiques, il convient d'étendre les conventions de priorité établies précédemment.

Nous avons quatre classes d'opérateurs indiquées dans la liste qui suit, en partant de la plus prioritaire :

1. ()
2. not
3. * / div mod and
4. + - or xor
5. = <> < > <= >=

FIGURE 2.5 – Précédence des opérateurs

Dans chacune des classes, les opérateurs ont la même priorité. En cas d'égalité de priorité, les opérations correspondantes sont effectuées de gauche à droite. Les parenthèses peuvent servir à forcer la priorité. Voici, à titre d'exemple, comment s'exprimerait en Pascal l'expression "somme est comprise entre 10 et 35" :

```
(somme >= 10) and (somme <= 35)
```

Examinons deux autres exemples d'expressions booléennes :

```
age < date + 100
```

il s'agit ici de comparer le contenu de la variable age avec le résultat de date + 100. L'opérateur + a une plus grande priorité que l'opérateur <. De ce fait, cette expression booléenne est équivalente à

```
age < (date + 100)
```

Mais

```
age < 40 and revenu > 6000
```

cette expression provoquera un message d'erreur de la part du compilateur, car elle présente un conflit de types. En effet, la priorité de l'opérateur *and* étant plus élevée que celle des opérateurs relationnels < et >, elle sera interprétée comme

```
age < (40 and revenu) > 6000
```

qui n'est pas une expression correcte. Il faudra écrire

```
(age < 40) and (revenu > 6000)
```

2.6.5 Type caractère (char)

Ce type dénote un ensemble de caractères, fini et ordonné. Chacun de ces caractères peut être exprimé grâce à son code ASCII. Une variable de type caractère (char) peut contenir un seul caractère, généralement spécifié entre apostrophes. Comme nous le verrons plus loin, il est possible de constituer des suites de caractères appelées chaînes de caractères. Voici un petit programme qui met en évidence l'emploi des variables et constantes de type caractère :

```
const effe = 'F';
var lettre  : char;
    bip     : char;
begin
    bip := Chr(7);
    lettre := 'h';
end;
```

Dans cet exemple, la constante *effe* contient une lettre majuscule indiquée entre apostrophes et la variable *bip* contient le caractère dont le code ASCII est 7. Ce code correspond à l'émission d'un bref signal sonore par le haut-parleur de l'ordinateur. La fonction prédéfinie *Chr* permet de référencer tous les caractères, y compris certains caractères du code ASCII qui ne sont pas affichables. L'argument de cette fonction est précisément le code ASCII du caractère désiré. Pour connaître le code des caractères disponibles, il convient de se reporter au manuel de référence de l'ordinateur utilisé.

On peut aussi écrire un caractère en faisant précéder son code par le symbole #. Cette notation est équivalente à la fonction *Chr* et permet, par exemple, d'incorporer des caractères particuliers dans une chaîne de caractères.

```
#27'G'
```

cette suite de caractères comprend le caractère correspondant à la touche <Esc>, suivi du caractère G. ;

```
if c = #13 then...
```

Cette instruction permet de déterminer si la variable c (de type caractère) contient le caractère correspondant à la touche <Return>. On aurait également pu écrire :

```
if c = chr(13) then...
```

2.6.6 Les chaînes de caractères (String)

2.6.6.1 Définitions

- Une chaîne de caractères est un ensemble de caractères délimité par des apostrophes (’). Elles se définissent par **String**, **AnsiString** ou **WideString**.
- Le type *String* peut faire référence à *ShortString* ou *AnsiString*, selon la bascule *{$H}*. Si la bascule est off (*{$H-}*) alors toute déclaration String va définir un ShortString. Sa taille sera de 255 caractères, si elle n’est pas spécifiée autrement. Si la bascule est on (*{$H+}*), String sans spécifier la taille va définir un AnsiString, sinon un ShortString avec une taille spécifiée.
  Par défaut, un String est un AnsiString.
- Une variable de type *WideString* ou *AnsiString* n’a pas de limite quant au nombre de caractères.
- On peut limiter la taille d’une chaîne en écrivant *String[n]* où $n \leq 255$.

2.6.6.2 Le traitement des chaînes de caractères

Remarque Seules, quelques fonctions ou procédures qui manipulent les chaînes seront traitées ici.

Concaténation (regroupement) de chaînes On regroupe deux ou plusieurs chaînes en utilisant l’opérateur **+** [4].
Exemple

```
ch1 := 'Bonjour ';
ch2 := 'tout le monde!';
ch3 := ch1 + ch2; // ch3 vaut 'Bonjour tout le monde!'
```

Longueur d’une chaîne La longueur d’une chaîne correspond à son nombre de caractères. On l’obtient grâce à **Length**.
Exemple

```
s := 'Bonjour tout le monde!';
nbCar := Length(s);   // nbCar vaut 22
```

Extraction de caractères

- Pour extraire **un seul** caractère d’une chaîne, on utilise *[position]* où position est l’indice du caractère souhaité. position commence à un : le premier caractère de la chaîne s s’obtient en écrivant s[1], le second s[2], $\cdots$.
  Exemple

```
s := 'Bonjour tout le monde!';
car := s[6];    // car vaut 'u'
```

4. **Concat** permet aussi d’ajouter une chaîne à une autre

- Pour extraire **plusieurs** caractères d'une chaîne, c'est-à-dire extraire une sous-chaîne, on utilise ***copy****(chaine, debut, combien)*.
  Exemples

```
s := 'Bonjour tout le monde!';
ch := copy(s,6,5);    // ch vaut 'ur to'
```

Comparaison de chaînes Pour déterminer l'égalité de deux chaînes en tenant compte de la casse des caractères, on utilise l'opérateur **=**.
Exemples

```
s1 := 'Bonjour';
s2 := 'Bonjour';
//s1 = s2 est vrai (true)
s1 := 'Bonjour';
s2 := 'bonjour';
// s1 = s2 est faux (false)
```

Changement de la casse des caractères

- **UpCase** convertit tous les caractères d'une chaîne en majuscules.
  Exemple

```
s1 := 'Bonjour';
s2 := UpCase(s1); // s2 vaut BONJOUR
```

  UpCase permet aussi de mettre un caractère en majuscule ; elle retourne alors un caractère.

- **LowerCase** convertit tous les caractères d'une chaîne en minuscules. Les caractères non alphabétiques ne sont pas affectés par l'opération.
  Exemple

```
s1 := 'Bonjour';
s2 := LowerCase(s1); // s2 vaut bonjour
```

  LowerCase permet aussi de mettre un caractère en minuscule ; elle retourne alors un caractère.

Recherche dans des chaînes **Pos** recherche la première occurrence d'un caractère ou d'une chaîne dans une chaîne. Si le caractère ou la chaîne n'est pas inclus dans la chaîne, le résultat de Pos est 0 sinon le résultat est le premier endroit où se trouve le caractère ou la chaîne (commence à un).
Exemples

```
s1 := 'Bonjour';
car := 'o';
endroit := Pos(car,s1); // endroit vaut 2

s1 := 'Bonjour';
s2 := 'njo';
endroit := Pos(s2,s1);  // endroit vaut 3

s1 := 'Bonjour';
s2 := 'xy';
endroit := Pos(s2,s1);  // endroit vaut 0 car xy n'est pas dans Bonjour
```

Effacement de caractères dans une chaîne Pour effacer des caractères d'une chaîne, on utilise la procédure ***Delete****(chaine, debut, combien).*
Exemple

```
s := 'Bonjour tout le monde!';
Delete(s,5,3); // s vaut 'Bonj tout le monde!'
```

Insertion d'une chaîne dans une autre Pour insérer une chaîne dans une autre, on utilise la procédure ***Insert****(chaineAInserer, cible, position).*
Exemple

```
Cible := '12345678';
Insert('+++', Cible, 3);    // Cible vaut '12+++345678'
```

Créer une chaîne formée du même caractère Pour créer une chaîne formée de plusieurs fois le même caractère, on utilise la fonction ***StringOfChar****(car, nombre).*
Exemple

```
etoile := StringOfChar('*', 10); // **********
```

Transformation en numérique et inversement

- La procédure **Str** transforme une valeur numérique en chaîne. Par exemple,

  Str(25, ch1) met '25' (une chaîne) dans ch1
  Str(*pi*, ch2) met la valeur de *pi* dans une chaîne

- La procédure **Val** transforme une chaîne en valeur numérique, si possible. Par exemple,

  Val('25',i,code) met 25 (entier) dans i et *code* sera égal à 0
  Val('bonjour',i,code) n'est pas possible, *code* **n'est pas nul** et i est indéfini
  Val('3.14',x,code) met une approximation de π dans la variable x (supposée réelle)

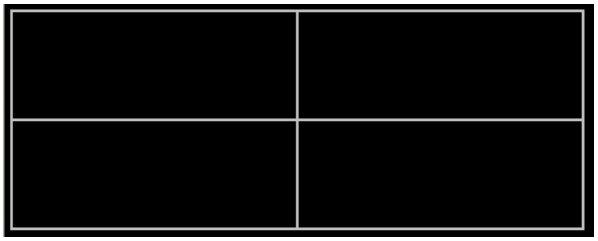
- La fonction **Chr** convertit un code ASCII en caractère. Par exemple,

  ```
  write(chr(170));   // Caractère ┐
  ```

Chr(169) ┌	Chr(170) ┐	Chr(179) │	Chr(180) ┤	Chr(181) ╡
Chr(182) ╢	Chr(183) ╖	Chr(184) ╕	Chr(185) ╣	Chr(186) ║
Chr(187) ╗	Chr(188) ╝	Chr(189) ╜	Chr(190) ╛	Chr(191) ┐
Chr(192) └	Chr(193) ┴	Chr(194) ┬	Chr(195) ├	Chr(196) −
Chr(197) ┼	Chr(198) ╞	Chr(199) ╟	Chr(200) ╚	Chr(201) ╔
Chr(202) ╩	Chr(203) ╦	Chr(204) ╠	Chr(205) =	Chr(206) ╬
Chr(207) ╧	Chr(208) ╨	Chr(209) ╤	Chr(210) ╥	Chr(211) ╙
Chr(212) ╘	Chr(213) ╒	Chr(214) ╓	Chr(215) ╫	Chr(216) ╪
Chr(217) ┘	Chr(218) ┌	Chr(219) ▉		

FIGURE 2.6 – Quelques codes ASCII

Voici le code (assez pénible) pour afficher le cadre suivant



```
program Cadre;
var ligneHaut : string;
    ligneBlancs : string;
    ligneMilieu : string;
    ligneBas :string;
begin
    ligneHaut   := chr(218) + StringOfChar(chr(196),20) + chr(194) +
                   StringOfChar(chr(196),20) + chr(191);
    ligneBlancs := chr(179) + StringOfChar(' ',20) + chr(179) +
                   StringOfChar(' ',20) + chr(179);
    ligneMilieu := chr(195) + StringOfChar(chr(196),20) + chr(197) +
                   StringOfChar(chr(196),20) + chr(180);
    ligneBas    := chr(192) + StringOfChar(chr(196),20) + chr(193) +
                   StringOfChar(chr(196),20) + chr(217);

    writeln(ligneHaut);
    writeln(ligneBlancs);
    writeln(ligneBlancs);
    writeln(ligneBlancs);
    writeln(ligneMilieu);
    writeln(ligneBlancs);
    writeln(ligneBlancs);
    writeln(ligneBlancs);
    writeln(ligneBas);

    readln;
end.
```

Pour faciliter le travail, l'utilisation de constantes pourrait être utile. Par exemple,

```
const CGH = chr(218);     // Coin Gauche Haut
      CDH = chr(191);     // Coin Droit Haut
      CBG = chr(192);     // Coin Bas Gauche
      CBD = chr(217);     // Coin Bas Droit
      VERT = chr(179);    // Verticale
      VD = chr(195;       // Vertical Droite
      CG = chr(180);      // Vertical Gauche
      HORIZ = chr(196);   // Horizontale
      HB = chr(194);      // Horizontal Bas
      HH = chr(193;       // Horizontal Haut
      CROIX = chr(197);   // Croix
```

Le code s'écrira :

```
ligneHaut := CGH + StringOfChar(HORIZ,20) + HB + StringOfChar(HORIZ,20) + CDH;
ligneBlancs := VERT + StringOfChar(' ',20) + VERT + StringOfChar(' ',20) + VERT;
                    ⋮
```

Quelques caractères accentués

chr(133)	à	chr(131)	â	chr(132)	ä		
chr(130)	é	chr(138)	è	chr(136)	ê	chr(137)	ë
chr(140)	î	chr(139)	ï				
chr(147)	ô						
chr(150)	û	chr(151)	ù	chr(154)	ü		
chr(135)	ç						

FIGURE 2.7 – Quelques codes ASCII : les caractères accentués

2.6.6.3 Autres fonctions ou procédures qui manipulent les chaînes

Toutes les fonctions ou procédures traitées dans ce point font partie de l'unité *System* (elles s'utilisent sans *uses* explicite). Il en existe encore une multitude d'autres, définies dans les unités *SysUtils* et *StrUtils* (*uses* est alors nécessaire).

On peut citer dans StrUtils :

- *RightStr* qui extrait des caractères à partir de la fin

  ```
  writeln (RightStr('abcdefghijklmnopqrstuvwxyz', 10)); // qrstuvwxyz
  ```

- *NPos* qui calcule la position de la n^e occurrence d'une sous-chaîne dans une chaîne.

  ```
  writeln (NPos('ab','abcdeabceabxyz', 3)); // 10
  ```

- *PadLeft* qui ajoute des espaces au début d'une chaîne jusqu'à ce qu'une certaine longueur soit atteinte.

  ```
  writeln (PadLeft('1234', 6)); // deux espaces suivis de 1234
  ```

- *PadRight* qui ajoute des espaces à la fin d'une chaîne jusqu'à ce qu'une certaine longueur soit atteinte.

```
writeln (PadRight('1234', 6));    // 1234 suivi de deux espaces
```

- *PosEx* qui recherche la position d'une sous-chaîne dans une chaîne en commençant à un position spécifiée.

```
writeln (PosEx('ab','abcdeabceabxyz', 8));    // 10
```

- *ReverseString* qui inverse une chaîne de caractères.

```
writeln (ReverseString('abcde'));    // 'edcba'
```

- *WordCount* qui compte le nombre de mots d'une chaine.

```
writeln (WordCount('Ceci est une chaine',StdWordDelims));    // 4
```

Remarque `StdWordDelims = [#0..' ', ',', '.', ';', '/', '\', ':', '"', '''', '`'] + Brackets`

- *RPos* qui recherche la dernière position d'une sous-chaîne dans une chaîne.

```
writeln (RPos('ab','abcdeabceabxyz')); // 10
```

- *RPos* qui recherche la dernière position d'une sous-chaîne dans une chaîne.

```
writeln (RPos('ab','abcdeabceabxyz')); // 10
```

- ⋮

On peut citer dans SysUtils :

- *StringReplace* qui remplace les occurrences d'une sous-chaîne par une autre dans une chaîne.

```
writeln(StringReplace('abcdeabceabxyz','ab','mno',[rfReplaceAll]));
  // 'mnocdemnocemnoxyz'
```

- *TrimLeft* qui supprime les espaces au début d'une chaîne.

```
writeln(TrimLeft('␣␣Des␣espaces␣␣')); ⟹ 'Des␣espaces␣␣'
```

- *TrimRight* qui supprime les espaces à la fin d'une chaîne.

```
writeln(TrimRight('␣␣Des␣espaces␣␣')); ⟹ '␣␣Des␣espaces'
```

- ⋮

2.6.7 Autres types de données

D'autres types de données existent en Pascal. Il s'agit, entre autres, du type énuméré, du type intervalle, du type ensemble, du type pointeur, du type tableau, du type fichier. Ces types seront étudiés plus tard.

CHAPITRE 3

LES INSTRUCTIONS EN PASCAL

3.1 Les commentaires

Les commentaires ont été étudiées à la page 21.

Résumé on écrit des commentaires dans un programme en les entourant de (* *) ou { }.
Pour les commentaires d'une seule ligne, on peut utiliser //.
Les commentaires sont ignorés par le compilateur.

3.2 L'affectation

Cf. page 22.

Résumé il est possible d'affecter une valeur à une variable grâce à :=
Par exemple,

```
i  := 10;
ch := 'Quel est votre nom ? ';
x  := cos (pi * sqrt(2)) / arctan (10);
```

3.3 Lecture et écriture

3.3.1 Lecture du contenu d'une variable au clavier

Le but est de mettre une valeur dans une variable grâce au clavier. Autrement dit, ce que l'utilisateur va taper sur son clavier « va aller » dans la variable.
L'opération qui consiste à remplir le contenu d'une variable à partir du clavier s'appelle une **lecture**.

- Dans un algorithme, on écrira

```
lire v où v est une variable quelconque
```

- dans un organigramme, on représente la lecture par

- en Pascal, la lecture est accomplie avec l'instruction **readln**

```
readln(v);
```

3.3.2 Écriture du contenu d'une variable à l'écran

L'opération qui consiste à afficher la valeur d'une variable à l'écran s'appelle une **écriture**.

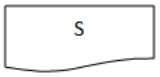
- Dans un algorithme, on écrira

```
écrire v où v est une variable quelconque
```

ou

```
affiche v
```

- dans un organigramme, on représente l'écriture par



- en Pascal, pour afficher la valeur d'une variable sur une console DOS, il faut écrire

```
write(expr1[,expr2...,]);
```

ou

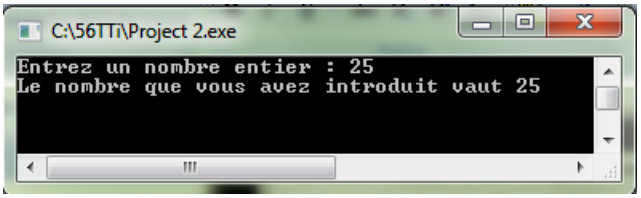
```
writeln(expr1[,expr2...,]);
```

où expr est une expression contenant une chaîne de caractères, une variable ou une constante. Dans la première expression, le curseur reste à la fin du texte affiché, dans la seconde, le curseur se place au début de la ligne suivante.

3.3.3 Exemple

```
program demo;
var i : integer;   // Déclaration de la variable
begin
    write('Entrez un nombre entier : '); // Affiche ce message
    readln(i);

    // Affichage d'un message suivi du contenu de la variable i
    writeln('Le nombre que vous avez introduit vaut ',i);

    // Pour "figer" l'affichage
    readln;
end.
```

Voici le résultat :



3.3.4 Exercices

1. Afficher bonjour à l'écran.
2. Afficher une carte de visite à l'écran.
3. Calculer la somme et le produit de deux nombres réels.
4. Calculer l'aire et le périmètre d'un rectangle connaissant sa largeur et sa hauteur.
5. Calculer le volume de la sphère.

$$Volume = \frac{4}{3}\pi Rayon^3$$

   On obtient la valeur de π en écrivant pi.
6. Échanger le contenu de deux variables.
7. Lire deux entiers, afficher le quotient et le reste de la division.
8. Lire une température donnée en degrés Celsius et l'afficher en degrés Fahrenheit.

$$Fahrenheit = \frac{9}{5}Celsius + 32 \qquad \Longleftrightarrow \qquad Celcius = \frac{5}{9}(Fahrenheit - 32)$$

9. Lire une température donnée en degrés Fahrenheit et la transformer en degrés Celcius.
10. Lire un nombre de secondes et le traduire sous la forme d'heures, minutes et secondes.
    (4354 secondes = 1 heure 12 minutes 34 secondes).
11. Transformer un angle donné en radians en degrés.

$$degre = \frac{180}{\pi}radian \qquad \Longleftrightarrow \qquad radian = \frac{\pi}{180}degre$$

12. Transformer un angle donné en degrés en radians.
13. Lire un nombre de degrés, minutes, secondes et le transformer en radian.
    (Il faut d'abord réaliser le passage Degrés Minutes Secondes en degrés)
14. Lire un angle en radian et le transformer en degrés, minutes, secondes et millièmes de seconde.
15. Lire un nombre de jours et le transformer sous la forme d'années, mois et jours en supposant qu'un mois contiendra toujours trente jours.
16. Calculer la résistance (électrique) R de trois résistances en parallèle R_1, R_2, R_3,.

$$\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3}$$

17. Calculer le périmètre et l'aire d'un triangle dont on connaît la longueur des 3 côtés.

$$Aire = \sqrt{\frac{p}{2}(\frac{p}{2} - cote_1)(\frac{p}{2} - cote_2)(\frac{p}{2} - cote_3)}$$

où p est le périmètre du triangle ($p = cote_1 + cote_2 + cote_3$).
On obtient la racine carrée de x ($\sqrt{x}$) en écrivant $sqrt(x)$ [1].

18. Lire trois chaînes de caractères et les afficher en une seule chaîne.
19. Calculer le temps écoulé entre deux événements.

3.4 Les tests

3.4.1 La structure alternative

En français, une alternative est un choix entre **deux possibilités**. La structure alternative peut s'écrire :

si *condition* **alors**
| action1(s)
sinon
| action2(s)
fin

Signification : si la condition est vérifiée, les actions *action1(s)* seront exécutées ; dans le cas contraire, les actions *action2(s)* seront exécutées.

Dans les organigrammes, le losange représente une alternative

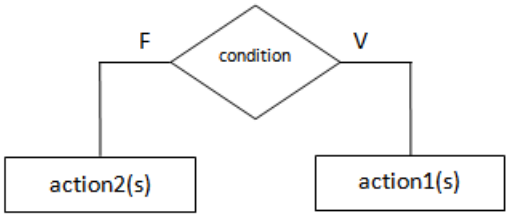


FIGURE 3.1 – L'alternative

En Pascal, on écrira :

```
if condition then
begin
   instruction(s) séparées par des ;
end
else
begin
   instruction(s) séparées par des ;
end;
```

1. Le carré de x peut s'obtenir en écrivant $sqr(x)$

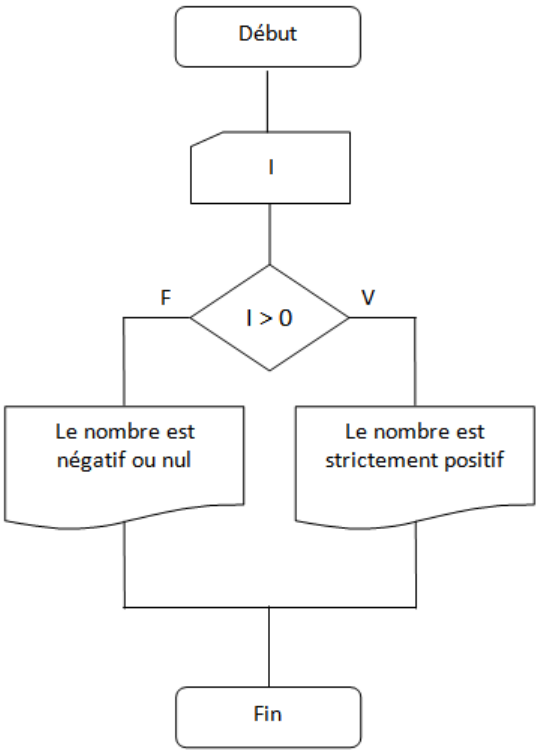
Remarques : Il est *interdit* de mettre un point-virgule avant ELSE.

Les mots BEGIN et END ne sont nécessaires que s'il y a plusieurs instructions.

La partie ELSE est facultative.

Exemple : lire un nombre entier et dire s'il est strictement positif.

lire I
si $I > 0$ **alors**
| **écrire** 'Le nombre est strictement positif'
sinon
| **écrire** 'Le nombre est négatif ou nul'
fin




```
program test;
var i : integer;   // Déclaration de la variable
begin
    write('Entrez un nombre entier : ');
    readln(i);   // Lecture au clavier

    if i > 0 then
        writeln('Le nombre est strictement positif')
    else
        writeln('Le nombre est negatif ou nul');

    readln;
end.
```

3.4.2 Le choix multiple

Supposons qu'on veuille afficher le jour de la semaine connaissant son numéro (1=dimanche, $\cdots$, 7=samedi).

On peut écrire en utilisant la structure alternative :

```
if numJour = 1 then
   write('Dimanche')
else if numJour = 2 then
   write('Lundi')
else if numJour = 3 then
   write('Mardi')
else if numJour = 4 then
   write('Mercredi')
else if numJour = 5 then
   write('Jeudi')
else if numJour = 6 then
   write('Vendredi')
else if numJour = 7 then
   write('Samedi');
```

ou encore

```
if numJour = 1 then
   write('Dimanche');
if numJour = 2 then
   write('Lundi');
if numJour = 3 then
   write('Mardi');
if numJour = 4 then
   write('Mercredi');
if numJour = 5 then
   write('Jeudi');
if numJour = 6 then
   write('Vendredi');
if numJour = 7 then
   write('Samedi');
```

Le mieux est certainement d'utiliser le choix multiple :

suivant *numJour* **faire**
  cas où *1* **faire** Afficher 'Dimanche';
  cas où *2* **faire** Afficher 'Lundi';
  cas où *3* **faire** Afficher 'Mardi';
  cas où *4* **faire** Afficher 'Mercredi';
  cas où *5* **faire** Afficher 'Jeudi';
  cas où *6* **faire** Afficher 'Vendredi';
  cas où *7* **faire** Afficher 'Samedi';
fin

Dans les organigrammes, on peut représenter un choix multiple de la manière suivante :

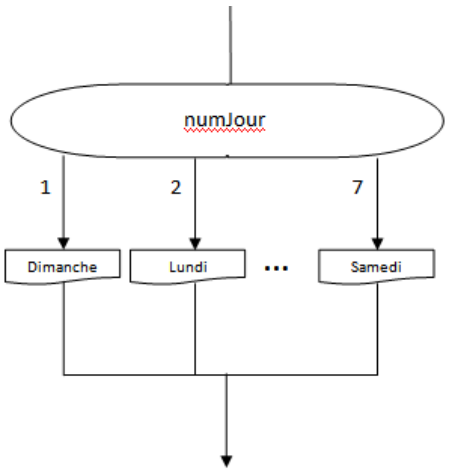


FIGURE 3.2 – Le choix multiple

En Pascal,

```
case numJour of
    1 : write ('Dimanche');
    2 : write ('Lundi');
    3 : write ('Mardi');
    4 : write ('Mercredi');
    5 : write ('Jeudi');
    6 : write ('Vendredi');
    7 : write ('Samedi');
end;
```

Cette nouvelle structure agit en fait à la manière d'un test à multiples branches. Pour chaque branche l'égalité entre la valeur de numJour et la constante correspondante est vérifiée. Si effectivement la valeur de numJour équivaut à une des constantes, l'instruction ou le bloc d'instructions (pour rappel, un bloc d'instructions est un ensemble d'instructions séparés par les mots Begin et End) correspondant sont exécutés. Si la valeur du sélecteur ne correspond à aucune des constantes indiquées l'exécution du programme se poursuit après la structure sélective.

Une branche supplémentaire indiquée par un **else** permet l'exécution d'une instruction ou d'un bloc d'instructions au cas où la valeur du sélecteur ne correspond à aucune des constantes spécifiées. L'exemple qui suit illustre cette possibilité :

```
case numJour of
   1 : write ('Dimanche');
   2 : write ('Lundi');
   3 : write ('Mardi');
   4 : write ('Mercredi');
   5 : write ('Jeudi');
   6 : write ('Vendredi');
   7 : write ('Samedi');
else
   write ('Erreur');
end;
```

Remarquons qu'avant le mot réservé *else* d'une structure case peut se trouver un point virgule.

Dans chaque branche d'une instruction *case*, on peut indiquer plus d'une valeur (2,4,7), et même un intervalle de valeurs (2..7). Par exemple,

```
case resultat of
   1..5: Valeur := 'Low';
   6..9: Valeur := 'High';
   0, 10..99: Valeur := 'Hors limite';
else
   Valeur := '';
end;
```

S'il y a plusieurs instructions dans une branche du *case*, il faut les entourer de **begin** et **end**.

```
case resultat of
   1 : begin
         resultat := 1;
         writeln('Resultat = ', resultat);
       end;
   ⋮
```

Comme d'habitude, le langage (Pascal) autorise l'utilisation de begin et end même s'il n'y a qu'une seule instruction.

Voir la solution de l'exercice en page 174 pour un exemple plus détaillé d'un choix multiple.

3.4.3 Exercices

1. Afficher le plus grand et le plus petit de deux nombres réels.
2. Calculer la racine carrée si cette opération est possible.
3. Résoudre l'équation du second degré.

   Lire a,b,c
   $\delta = b^2 - 4ac$
   si $\delta > 0$ **alors**
   | $racine1 = \frac{-b+\sqrt{\delta}}{2a}$
   | $racine2 = \frac{-b-\sqrt{\delta}}{2a}$
   sinon si $\delta = 0$ **alors**
   | $racine1 = racine2 = \frac{-b}{2a}$
   sinon
   | Pas de racine
   fin

4. Lire trois nombres, en écrire le plus petit, le plus grand.
5. Résoudre le même problème que précédemment mais avec six nombres.
6. Déterminer si une année A est bissextile. Une année A est bissextile si A est multiple de 4, toutefois, les années séculaires ne sont bissextiles que si A est multiple de 400.
7. On appelle nombre d'Armstrong un nombre de trois chiffres tel que sa valeur égale la somme des cubes de ses chiffres. Par exemple,

   $$153 = 1^3 + 5^3 + 3^3$$

   Lire un nombre et afficher si c'est un nombre d'Armstrong (les nombres d'Armstrong sont 153, 370, 371 et 407) ;
8. Déterminer si un triangle est rectangle, connaissant la longueur des trois côtés. Un triangle est rectangle si $C^2 = A^2 + B^2$.
   A, B, C sont les trois côtés du triangle, C étant l'hypoténuse.
9. Calculer le prix à payer si une réduction de 5% est accordée sur la partie qui dépasse 100 €.
10. Afficher la position de la droite $ax + by + c = 0$ par rapport aux axes (parallèle à l'axe X, parallèle à l'axe Y, oblique).
11. Lire un caractère, le programme teste s'il s'agit d'une lettre majuscule, si oui, il renvoie cette lettre en minuscule, sinon il renvoie un message d'erreur.
12. Lire deux chaînes de caractères et afficher à partir de quel endroit on retrouve la première dans la seconde ; si elle ne s'y trouve pas, afficher "La première chaîne n'est pas dans la seconde".
13. Lire un caractère, déterminer si ce caractère est une lettre, un chiffre ou un autre caractère.
14. Lire une chaîne de caractères, écrire si elle commence ou se termine par une consonne ou une voyelle.

15. Lire deux réels et un symbole au clavier.
    - si le symbole vaut '+', on additionne les deux réels,
    - si le symbole vaut '-', on soustrait le second réel du premier,
    - si le symbole vaut '*' ou '.', on multiplie les deux réels,
    - si le symbole vaut '/' ou ' :', on divise le premier réel par le second,
    - si le symbole ne vaut aucune de ces valeurs, le programme indique qu'il y a eu une erreur
16. Lire deux chaînes de caractères et supprimer, si elle s'y trouve, la seconde de la première.
17. Lire le nom d'un fichier (une chaîne). Le programme vérifie que celui-ci possède l'extension « PAS ».

    Indication Une extension n'a pas toujours trois caractères mais est toujours précédées de ".".
18. Lire une expression sous forme de chaîne et l'évaluer.
    Exemples :
    - 8+9
    - 187 / 54
    - 24 div 7
    - 24 mod 7

    Simplification On suppose qu'on évalue une expression simple, c'est-à-dire deux opérandes entières et un opérateur.
19. Déterminer le jour de la semaine correspondant à une date donnée. Les jours de la semaine sont codifiés de la manière suivante :

    $0 \longrightarrow$ dimanche, $1 \longrightarrow$ lundi, $2 \longrightarrow$ mardi, $\cdots$, $6 \longrightarrow$ samedi.

    Le numéro de jour de la semaine est fourni par la valeur de

    $(N + AN + J + 5 \times S)$ mod 7 où
    - S indique les deux premiers chiffres du millésime,
    - AN les deux derniers chiffres du millésime pour tous les mois sauf janvier et février, auquel cas, il faut diminuer ce nombre d'une unité,
    - J représente le numéro du jour dans le mois,
    - N est fourni par la relation

      $$N = \lfloor \tfrac{13 \times M - 1}{5} \rfloor + \lfloor \tfrac{S}{4} \rfloor + \lfloor \tfrac{AN}{4} \rfloor$$

      Pour janvier et février, M vaut respectivement 11 et 12 et pour les autres mois, M est le numéro du mois diminué de 2 unités.

3.5 Les structures répétitives ou boucles

Il est souvent nécessaire d'indiquer qu'un ensemble d'instructions doit être exécuté plusieurs fois. Le langage Pascal contient trois méthodes pour répéter des instructions. Il est possible d'exécuter des instructions

1. un certain nombre de fois (*For*),
2. tant qu'une condition est vérifiée (*While*) ou
3. jusqu'à ce qu'une condition soit vérifiée (*Repeat*).

3.5.1 La boucle while (Tant Que)

La boucle *While* dit à l'ordinateur de répéter des actions tant qu'une certaine condition est vérifiée. La structure est la suivante :

tant que *condition* **faire**
| action(s)
fin

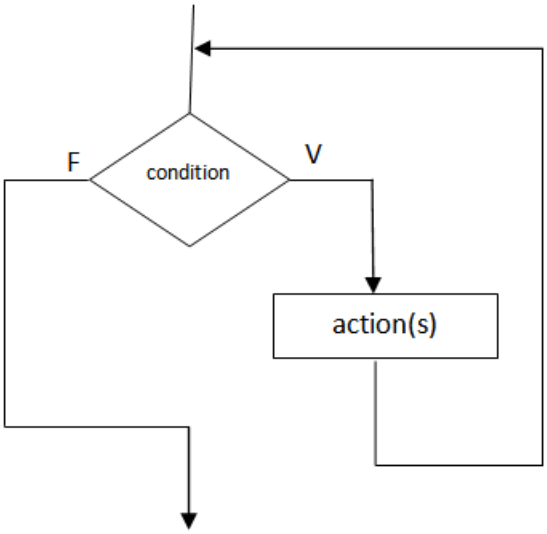


FIGURE 3.3 – La boucle While

la traduction en Pascal est :

```
while condition do
begin
   instruction(s)
end;
```

Remarques :

- L'ordinateur exécute les instructions indiquées entre BEGIN et END tant que la condition est **vraie**.
- BEGIN et END sont facultatifs s'il n'y a qu'une seule instruction à exécuter.

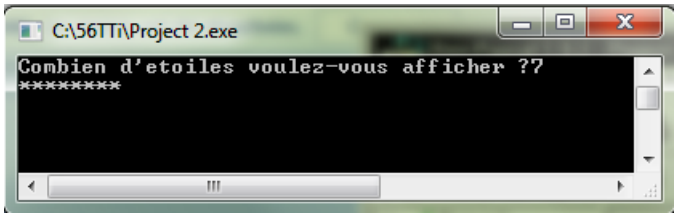
Exemple 1 Lire un nombre n au clavier. Afficher une ligne de n « * »
Ainsi,

- si le nombre entré au clavier est 2, le programme doit afficher **
- si le nombre entré au clavier est 5, le programme doit afficher *****
- si le nombre entré au clavier est 10, le programme doit afficher **********

Lire N
$Etoile \leftarrow 0$ *// Etoile est le nombre d'étoiles déjà affiché*

N étoiles ont-elles déjà été affichées ?
tant que $Etoile \leq N$ **faire**
  Afficher *
  $Etoile \leftarrow Etoile + 1$ *// Une étoile vient d'être affichée*
fin

```
1  program Etoiles;
2  (*******************************************************
3   * Lire (saisir) un nombre n au clavier                *
4   * Afficher une ligne de n etoiles                     *
5   *******************************************************)
6  var    n : integer;    // Le nombre d'étoiles à afficher
7      etoile : integer; // Le nombre d'étoiles déjà affichées
8  begin
9      //Saisie du nombre d'étoiles à afficher
10     Write('Combien d''etoiles voulez-vous afficher ?');
11     Readln(n);
12
13     // Affichage des étoiles
14     etoile := 0;
15     while etoile <= n do
16     begin
17         Write('*');     // Pas Writeln pour éviter le retour à la ligne
18         etoile += 1;    // ou etoile := etoile + 1
19     end;
20
21     Readln;
22 end.
```



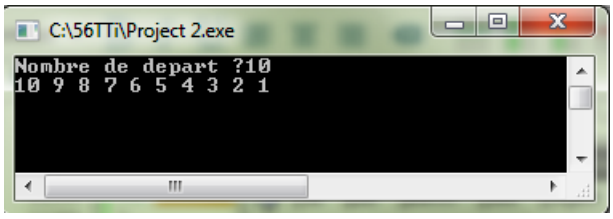
Exemple 2 Lire (saisir) un nombre n au clavier. Afficher les nombres de n jusqu'à 1 en diminuant de 1 à chaque fois
Ainsi,

- si le nombre entré au clavier est 5, le programme doit afficher 5 4 3 2 1
- si le nombre entré au clavier est 8, le programme doit afficher 8 7 6 5 4 3 2 1

Lire N
$I \leftarrow N$ *// Au départ, le premier nombre à afficher vaut N*

Est-on arrivé à 1 ?
tant que $I \geq 1$ **faire**
  Afficher I
  $I \leftarrow I - 1$ *// On diminue I de 1*
fin

```
1  program CompteARebours;
2  (*************************************************************
3     Lire (saisir) un nombre n au clavier. Afficher les
4     nombres de n jusqu'à 1 en diminuant de 1 à chaque fois.
5   *************************************************************)
6  var n : integer;   // Le nombre à partir duquel le compte à rebours commence
7     i : integer;    // Variable de travail
8  begin
9     // Saisie du nombre
10    Write('Nombre de depart ?');
11    Readln(n);
12
13    i := n;
14    // Affichage du compte à rebours
15    while i >= 1 do
16    begin
17       Write(i,' '); //Pas writeln pour éviter le retour à la ligne
18       i -= 1;   // ou i := i-1
19    end;
20
21    Readln;
22 end.
```



Questions à se poser Pour écrire une boucle, il faut se poser les questions suivantes.

1. Que doit-on écrire avant que la boucle ne commence ? Cette étape est l'**initialisation**.
2. Que doit faire la boucle à chaque étape ? Cette étape est le **traitement**.
3. Quand la boucle s'arrête-t-elle ou quelle est la **condition d'arrêt** ?

3.5.2 La boucle repeat (répéter … jusqu'à)

La boucle Repeat dit à l'ordinateur de répéter des actions jusqu'à ce qu'une certaine condition soit vérifiée. La structure est la suivante :

répéter
| instruction(s)
jusqu'à *condition d'arret*;

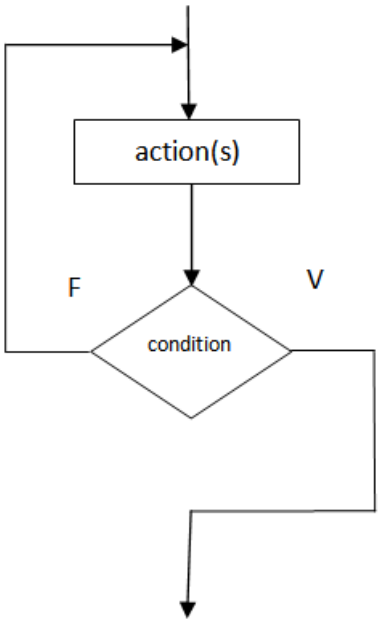


FIGURE 3.4 – La boucle Repeat

la traduction en Pascal est :

```
repeat
    instructions
until condition;
```

Remarques :

- L'ordinateur exécute les instructions indiquées entre REPEAT et UNTIL jusqu'à ce que la condition soit vraie. Autrement dit, **si la condition devient vraie, la boucle s'arrête**, ce qui est exactement l'opposé de la boucle WHILE.
- Du fait que le test d'arrêt est évalué en fin de boucle, la boucle REPEAT est toujours exécutée **au moins une fois**; ce qui n'est pas nécessairement le cas de la boucle WHILE.
- BEGIN et END sont inutiles puisque le début de la boucle est indiqué par REPEAT et la fin par UNTIL.

Exemple 1 Lire un nombre n au clavier. Afficher une ligne de n « * »

Lire N
$Etoile \leftarrow 0$ *// Etoile est le nombre d'étoiles déjà affiché*

N étoiles ont-elles déjà été affichées ?
répéter
  Afficher *
  $Etoile \leftarrow Etoile + 1$ *// Une étoile vient d'être affichée*
jusqu'à $Etoile > N$

```
1  program Etoiles;
2  (*******************************************************
3   * Lire (saisir) un nombre n au clavier                *
4   * Afficher une ligne de n etoiles                     *
5   *******************************************************)
6  var    n : integer;    // Le nombre d'étoiles à afficher
7      etoile : integer; // Le nombre d'étoiles déjà affichées
8  begin
9      //Saisie du nombre d'étoiles à afficher
10     Write('Combien d''etoiles voulez-vous afficher ?');
11     Readln(n);
12
13     // Affichage des étoiles
14     etoile := 0;
15     repeat
16         Write('*');     // Pas Writeln pour éviter le retour à la ligne
17         etoile += 1;   // ou etoile := etoile + 1
18     until etoile > n;
19
20     Readln;
21 end.
```

Exemple 2 Lire (saisir) un nombre n au clavier. Afficher les nombres de n jusqu'à 1 en diminuant de 1 à chaque fois.

Lire N
$I \leftarrow N$ *// Au départ, le premier nombre à afficher vaut N*
répéter
  Afficher I
  $I \leftarrow I - 1$ *// On diminue I de 1*
  // Est-ce fini ?
jusqu'à $I < 1$

```
1  program CompteARebours;
2  (*************************************************************
3     Lire (saisir) un nombre n au clavier. Afficher les
4     nombres de n jusqu'à 1 en diminuant de 1 à chaque fois.
5   *************************************************************)
6  var    n : integer;   // Le nombre à partir duquel le compte à rebours commence
7         i : integer;   // Variable de travail
8  begin
9     // Saisie du nombre
10    write('Nombre de depart ?');
11    readln(n);
12
13    i := n;
14    // Affichage du compte à rebours
15    repeat
16       write(i,' '); //Pas writeln pour éviter le retour à la ligne
17       i -= 1;   // ou i := i-1
18    until i < 1;
19
20    readln;
21 end.
```

3.5.3 La boucle for (pour)

La boucle for dit à l'ordinateur de répéter des actions un certain nombre de fois. La structure est la suivante :

pour $i \leftarrow debut$ ***à*** fin **faire**
| action(s)
fin

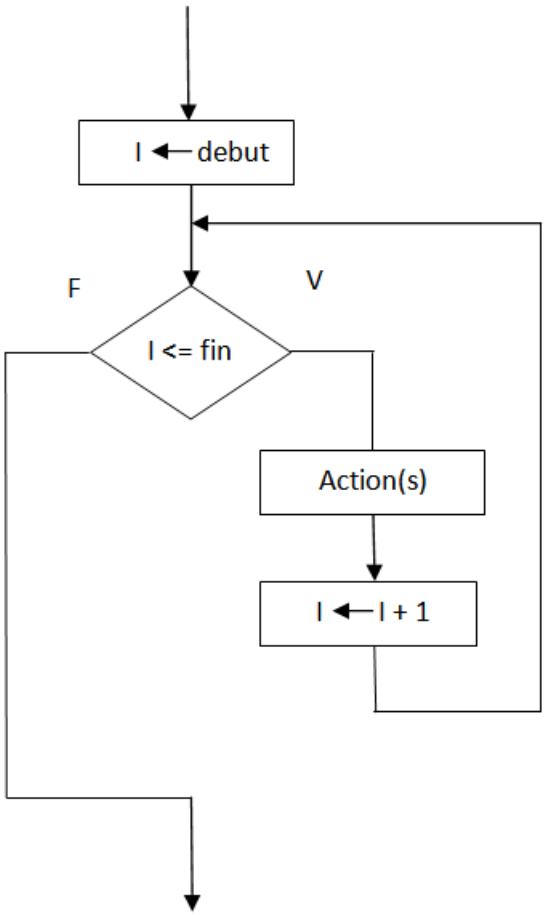


FIGURE 3.5 – La boucle For (to)

la traduction en Pascal est :

```
for I := debut to fin do
begin
   instruction(s)
end;
```

Remarques :

- Au départ, I est initialisé **automatiquement** à DEBUT. Puis, à chaque passage dans la boucle, I est augmenté **automatiquement** de un. La boucle s'arrête quand I dépasse FIN. La boucle est donc exécutée FIN-DEBUT+1 fois.

La boucle FOR est équivalente à une boucle WHILE avec deux choses qui se font automatiquement, à savoir l'initialisation de I à debut et l'augmentation de I de 1.

- BEGIN et END sont facultatifs s'il n'y a qu'une seule instruction à exécuter.
- La boucle FOR ne s'exécute pas si FIN < DEBUT.
- Il existe une variante de la boucle FOR qui permet de **décrémenter** la variable de contrôle (I) de un au lieu de l'augmenter (**Downto**) :

```
for I := debut downto fin do
begin
   instruction(s)
end;
```

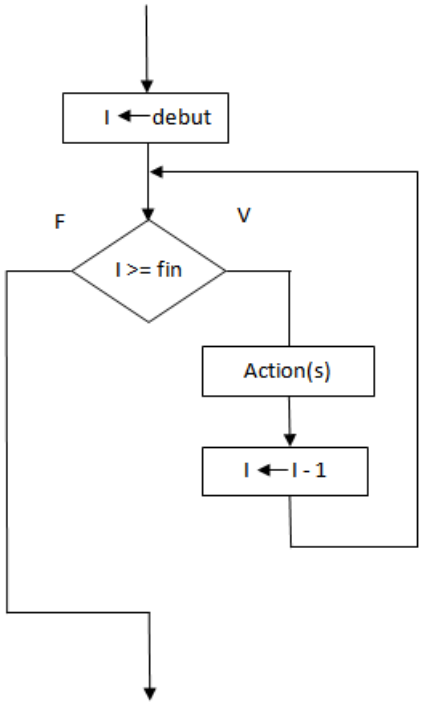


FIGURE 3.6 – La boucle For (downto)

Dans ce cours, la boucle FOR est la boucle la plus utile mais on ne peut l'utiliser que si le nombre de répétitions est **connu** !

Exemple 1 Lire un nombre n au clavier. Afficher une ligne de n « * »

Lire N

N étoiles ont-elles déjà été affichées ?
pour $etoile \leftarrow 1$ ***à*** N **faire**
| **Afficher** *
fin

```
1  program Etoiles;
2  (* Afficher une ligne de n etoiles *)
3  var    n : integer;    // Le nombre d'étoiles à afficher
4      etoile : integer; // Le nombre d'étoiles déjà affichées
5  begin
6      //Saisie du nombre d'étoiles à afficher
7      write('Combien d''etoiles voulez-vous afficher ?');
8      readln(n);
9
10     // On affiche une étoile N fois
11     for etoile := 1 to n do
12         write('*');
13
14     readln;
15 end.
```

Exemple 2 Afficher les nombres de n jusqu'à 1 en diminuant de 1 à chaque fois.

Lire N

pour $I \leftarrow N$ ***à*** 1 **faire**
| **Afficher** I
fin

```
1  program CompteARebours;
2  (* Afficher les nombres de n jusqu'à 1 *)
3  var    n : integer;   // Le nombre à partir duquel le compte à rebours commence
4         i : integer;   // Variable de travail
5  begin
6      // Saisie du nombre
7      write('Nombre de depart ?');
8      readln(n);
9
10     for i := n downto 1 do
11         write(i, ' ');
12
13     readln;
14 end.
```

3.6 Procédures Halt, Break, Continue, Exit

3.6.1 Halt

La procédure Halt provoque une fin anormale d'un programme et passe le contrôle au système d'exploitation.

3.6.2 Break

La procédure Break provoque l'interruption d'une boucle FOR, WHILE ou REPEAT.

L'exemple qui suit montre une procédure qui teste si un nombre nb est premier. Elle comporte une boucle for dans laquelle on détermine si le nombre nb est divisible par les nombres compris entre 2 et $nb - 1$. Dès que l'un de ces nombres divise nb, cela signifie que nb n'est pas premier. II est donc inutile de poursuivre l'exécution de la boucle, d'où l'utilisation de la procédure BREAK.

```
1  program premier;
2  var nb    : integer; // nombre à tester
3      i: integer; // indice de boucle
4      premier : boolean; // le nombre est-il premier ?
5  begin
6      write('Veuillez entrer un nombre entier : ');
7      readln(nb);
8
9      premier := true;   // On suppose que le nombre est premier
10     for i := 2 to nb - 1 do
11     begin
12         if nb mod i = 0 then
13         begin
14             premier := false; // nb n'est pas premier
15             break;
16         end;
17     end;
18     if premier = true then
19         write (nb, ' est un nombre premier')
20     else
21         write (nb, ' n''est pas un nombre premier');
22
23     readln;
24 end.
```

Remarque : la ligne 10 fait croire que la boucle va s'exécuter $(nb - 2)$ fois mais la ligne 15 stoppe la boucle. La condition d'arrêt de cette boucle est

- soit $i > nb - 1$;
- soit un diviseur a été trouvé.

Pour la lisibilité du programme, il aurait sans doute mieux valu utiliser une boucle WHILE

```
program premier;
var nb    : integer; // nombre à tester
    i: integer; // indice de boucle
    premier : boolean; // le nombre est-il premier ?
begin
    write('Veuillez entrer un nombre entier : ');
    readln(nb);

    premier := true;
    i := 2;
    while (i <= nb - 1) and (premier = true) do
    begin
        if nb mod i = 0 then
            premier := false; // nb n'est pas premier
        i := i + 1;
    end;

    if premier then    // ⟺ premier = true
        write (nb, ' est un nombre premier')
    else
        write (nb, ' n''est pas un nombre premier');

    readln;
end.
```

La condition d'arrêt est ici plus évidente.

3.6.3 Continue

L'appel à la procédure Continue provoque le passage du contrôle de l'exécution à l'itération suivante dans une instruction FOR, WHILE ou REPEAT.

3.6.4 Exit

La procédure Exit permet de quitter l'exécution de la procédure en cours. Si la procédure en cours correspond au programme principal, Exit termine l'exécution du programme.

3.6.5 Exercices

1. Calculer
   (a) $1 + 2 + 3 + \cdots + n$
   (b) $1 \times 2 \times 3 \times \cdots \times n$ $(n!)$

   où n est un entier positif.
2. Écrire un programme qui lit 10 nombres saisis au clavier et affiche le nombre de valeurs négatives lues.
3. Calculer le produit de nombres réels lus au clavier. On arrête dès qu'on a lu le réel 0.
4. Calculer la somme des
   (a) N premiers nombres pairs
   (b) N premiers nombres impairs

   où N est un entier positif lu au clavier.
5. Afficher tous les diviseurs d'un nombre entier donné.
6. Compter le nombre de diviseurs d'un nombre entier donné.
7. Écrire un programme qui affiche le nombre des entiers qui sont des multiples de 3 et inférieurs à un nombre n donné par l'utilisateur.
8. Calculer le plus grand et le plus petit parmi n nombres réels lus au clavier (n est aussi lu au clavier).
9. Même exercice que le précédent mais on arrête si le nombre réel à lire est 0.
10. Afficher le nombre de diviseurs de tous les entiers compris entre10 et 100.
11. Afficher le nombre de diviseurs de tous les entiers compris entre10 et 100. Le programme affichera encore celui qui a le plus de diviseurs.
12. Déterminer si un nombre entier est premier. Un nombre est premier s'il n'admet que deux diviseurs : 1 et lui-même. Par exemple, 23 est premier.
13. Déterminer si un nombre est parfait. Un entier est parfait s'il est égal à la somme de ses diviseurs, lui-même excepté (28 est un nombre parfait).
14. Calculer a^n si n est un entier positif.
15. Calculer a^n si n est un entier quelconque.
16. Afficher tous les nombres d'Armstrong (cf. exercice 7 page 45).
17. Afficher tous les nombres premiers compris entre 1 et 1000.
18. Afficher tous les nombres parfaits (cf. exercice 13) compris entre 1 et 10000 (il y en a quatre : 6, 28, 496, 8128).
19. Imprimer tous les couples (x,y) de nombres entiers compris entre -100 et 100 vérifiant la relation $9x - 4y = 35$
20. Imprimer tous les triplets (x,y,z) de nombres entiers compris entre 1 et 100 vérifiant la relation de Pythagore. La relation de Pythagore est $x^2 + y^2 = z^2$

21. Pour un capital C placé au taux t, calculer le total du capital et des intérêts obtenus au bout de n années (n est donné)
    (a) si le placement se fait à intérêt simple : $C_{final} = C_{depart}(1 + nt)$
    (b) si le placement se fait à intérêt composé : $C_{final} = C_{depart}(1 + t)^n$
22. Ecrire un programme qui gère le jeu de la "fourchette".
23. Calculer le nombre de 'e' d'une chaîne. (Il faut tenir compte des majuscules et des minuscules).
24. Calculer le nombre de voyelles, de consonnes et des autres caractères d'une chaîne.
25. Renverser une chaîne.
26. Tester si une chaîne est un palindrome. Un palindrome est une chaîne qui se lit de la même façon à l'endroit qu'à l'envers.
27. Compter le nombre de mots d'une chaîne. On suppose que les mots sont séparés par des espaces et des caractères de ponctuation.
28. Programmer le jeu du "pendu".
29. Programmer le jeu des "cinq lettres".

3.7 Utilisation du générateur aléatoire

3.7.1 Générer un nombre réel

Random écrit seul génère un nombre aléatoire **réel** compris dans l'étendue 0 <= X < 1 ($[0, 1[$)

```
function Random : Extended ;   // Il n'y a aucun paramètre
```

L'instruction

```
r := Random * (B–A) + A ;
```

génère donc un nombre réel compris entre A et B (B étant exclus). Par exemples,

- random * 20 génère un réel compris entre 0.0 et 20.0 ($[0, 20[$) ;
- random * 100 - 15 génère un réel compris entre -15 et 85 ($[-15, 85[$) ;
- random * 6 + 1 génère un réel compris entre 1 et 7 ($[1, 7[$) ;

3.7.2 Générer un nombre entier

```
function Random(const ARange: Integer): Integer;   // Il y a un paramètre
```

Quand un nombre entier est indiqué comme paramètre (ARange), la fonction Random renvoie un nombre aléatoire **entier** compris dans l'étendue 0 <= X < ARange-1.

Par exemples,
- random(20) génère un entier compris entre 0 et 19 ;
- random(100) + 15 génère un entier compris entre 15 et 114 ;
- random(6) + 1 génère un entier compris entre 1 et 6 ;

Pour générer un entier i compris entre A et B, il suffit d'écrire

```
i := Random (B–A+1) + A ;
```

3.7.3 Randomize

Randomize initialise le générateur interne de nombre aléatoire avec une valeur aléatoire (obtenue à partir de l'horloge du système).

```
procedure Randomize;
```

Exemple : Calculer la probabilité d'obtenir un 1 avec un dé

```
procedure TFiche.BoutonClick(Sender: TObject);
var nombreTirer    : integer;
    combienDeFois  : integer;
    combienDeUn : integer;
    probabilite : extended;
    i   : integer;
begin
    // Initialisations
    Randomize;
    combienDeFois := 1000000;   // On tente l'expérience 1 000 000 de fois
    combienDeUn := 0;

    // Calcul
    for i:=1 to combienDeFois do
    begin
        nombreTirer := Random(6) + 1; // Simulation d'un jet de dé (entre 1 et 6)
        if nombreTirer = 1 then // Tester si 1
            combienDeUn := combienDeUn + 1; //Ajout d'une unité au compteur de 1
    end;
    probabilite := combienDeUn / combienDeFois;
    writeln (probabilite:0:3);
end;
```

CHAPITRE 4

LES TABLEAUX

4.1 Les tableaux à une dimension

4.1.1 Définition et utilité

Si on veut utiliser huit variables entières dans un programme, une solution serait d'écrire

```
var i1,i2,i3,i4,i5,i6,i7,i8 : integer;
```

une autre est d'utiliser les tableaux.

Un tableau est une structure constituée d'un nombre déterminé de composants, tous du même type (les composants sont parfois appelés éléments). Les composants sont repérés à l'intérieur du tableau par un indice. On peut représenter schématiquement un tableau (appelé T) de 8 entiers de la manière suivante :

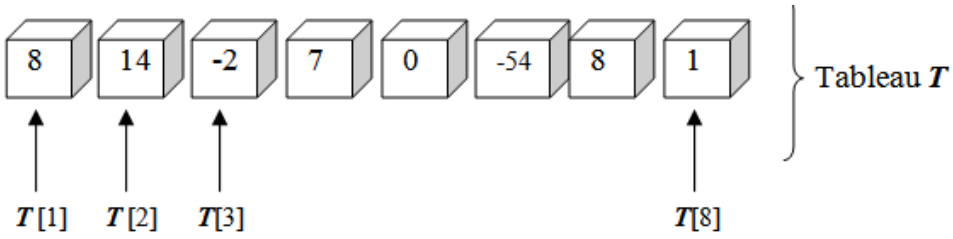


FIGURE 4.1 – Représentation d'un tableau

On déclare le tableau T en écrivant :

```
var T : array[1..8] of integer ;
```

On accède à un élément particulier en écrivant le nom du tableau suivi du numéro de l'élément. Par exemple,

```
T[1] ou T[5] ou T[i], ···
```

D'une manière plus générale, on déclare un tableau de la manière suivante :

```
var nom_de_variable : array[indice_début..indice_fin] of type_éléments;
```

Voici quelques exemples,

```
var   T1 : array[1..30] of integer;
      T2 : array[5..20] of real;
      T3 : array[-50..10] of char;
      T4 : array['A'..'Z'] of byte;
      T5 : array[1..100] of string;
      ...
```

- T1 est un tableau qui pourra contenir 30 entiers (T1[1], T1[2], ..., T1[30]) ;
- T2 est un tableau qui pourra contenir 16 réels (T2[5], T2[6], ..., T2[20]) ;
- T3 est un tableau qui pourra contenir 61 caractères (T3[-50], T3[-49], ..., T3[10]) ;
- T4 est un tableau qui pourra contenir 26 entiers (T4['A'], T4['B'], ..., T4['Z']) ;
- T5 est un tableau qui pourra contenir 100 chaînes (T5[1], T5[2], ..., T5[100]).

C'est au moment de la déclaration que la place d'un tableau est allouée en mémoire.

4.1.2 Exemples

1. Créer le tableau de la page précédente.

```
program ex_tableau_1;
var T : array[1..8] of integer;
begin
   T[1] := 8;
   T[2] := 14;
   T[3] := -2;
   T[4] := 7;
   T[5] := 0;
   T[6] := -54;
   T[7] := 8;
   T[8] := 1;
end.
```

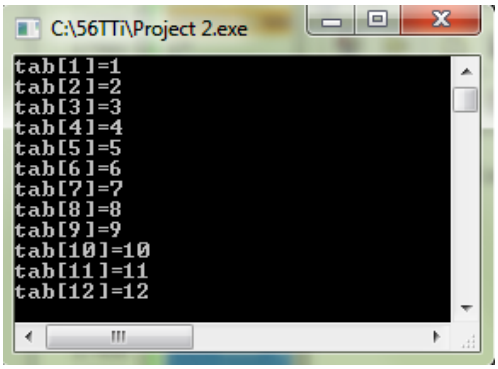
2. Créer un tableau de 12 entiers contenant 1 dans son premier élément, 2 dans le second, ..., 12 dans le dernier.

1	2	3	4	5	6	7	8	9	10	11	12
1	2	3	4	5	6	7	8	9	10	11	12

Dans cet exemple, l'indice est égal au contenu.

```
(*
   Créer un tableau de 12 entiers contenant 1 dans son premier élément,
   2 dans le second, ..., 12 dans le dernier.
*)
program tableau;
var    i : integer; // indice du tableau
       tab : array[1..12] of integer; // tableau de 12 entiers
begin
   // Remplissage du tableau
   for i := 1 to 12 do
   begin
      tab[i] := i; // L'indice est égal au contenu
   end;

   // Affichage du tableau
   for i := 1 to 12 do
   begin
      writeln('tab[',i,']=',tab[i]);
   end;

   readln;
end.
```

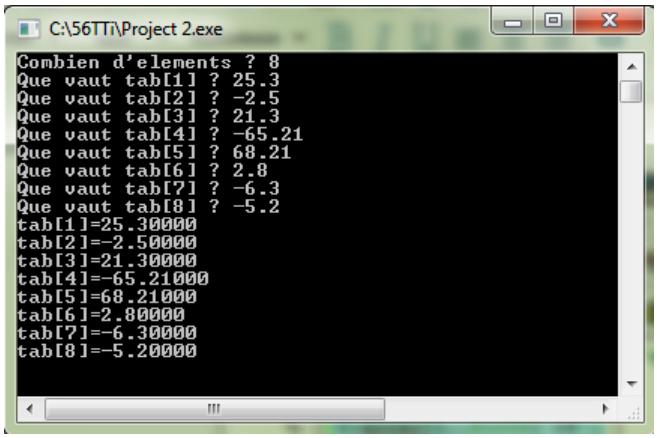
On obtient :



3. Lire un tableau de N réels au clavier. Afficher ensuite ce tableau.
   On suppose que N ne dépassera jamais 100.

   Remarques : on ne peut pas écrire "var t : array[1..N] of ..." car la valeur de N est inconnue au moment de la compilation. On écrira donc "var t :array[1..100] of ..." en sachant que l'utilisateur ne pourra entrer une valeur pour N qui soit supérieure à 100.
   Il existe aussi des **tableaux dynamiques** [1] dont le nombre d'éléments est fixé à l'exécution.

```
1  program tableau;
2  var    i:integer; //indice du tableau
3         tab:array [1..100] of real;    // Le tableau contient 100 éléments
4         taille : integer;//mais seuls les "taille" premiers éléments vont servir
5  begin
6     // Lecture du nombre d'éléments
7     write('Combien d''elements ? ');
8     readln (taille);
9     // Remplissage du tableau
10    for i := 1 to taille do
11    begin
12       write ('Que vaut tab[',i,'] ? ');
13       readln (tab[i]);
14    end;
15    // Affichage du tableau
16    for i := 1 to taille do
17    begin
18       writeln('tab[',i,']=',tab[i]:0:5);
19    end;
20    readln;
21 end.
```



Amélioration : Rien n'empêche l'utilisateur d'entrer une taille supérieure à 100. Le programme devrait donc interdire l'entrée d'un nombre supérieur à 100. Il suffit de remplacer les lignes 7 et 8 par

```
repeat
   write('Combien d''elements ? ');
   readln (taille);
until taille <= 100;
```

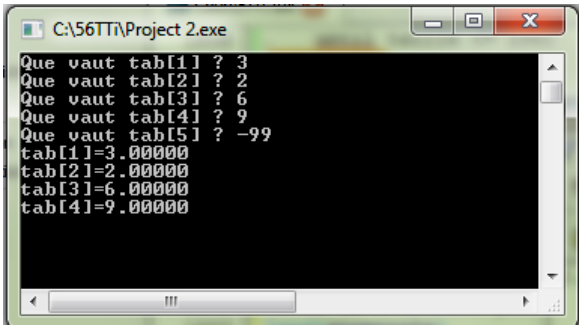
1. Les tableaux dynamiques seront étudiés en sixième

4. Lire un tableau de réels au clavier. La lecture s'arrête dès que l'utilisateur entre le nombre décimal -99.0. Afficher ensuite ce tableau.
   Cet exemple est identique au précédent sauf que la condition d'arrêt n'est plus "on a lu N éléments". Comme on ne sait pas d'avance combien d'éléments seront nécessaires,

   (a) on alloue un nombre d'éléments raisonnable, c'est-à-dire suffisamment grand mais pas inutilement grand ;

   (b) on n'utilise plus une boucle FOR mais une boucle REPEAT

```
program tableau;
(*
    Lire un tableau de réels au clavier.
    La lecture s'arrête dès que l'utilisateur entre
    le nombre décimal -99.0.
    Afficher ensuite ce tableau.
*)

var     i : integer; //indice du tableau
        tab : array[1..100] of real; // Taille raisonnable (100)
        taille_du_tableau : integer; // nombre d'éléments du tableau
begin
    // Remplissage du tableau
    i := 1;
    repeat
        write('Que vaut tab[',i,'] ? ');
        readln(tab[i]);
        inc(i);   // ou i+=1 ou i :=i+1
    until tab[i-1]= -99; // i-1 car on a incrémenté i

    taille_du_tableau := i - 2;
    // i-2 car on a incrémenté i et on ne prend pas le -99

    // Affichage du tableau
    for i := 1 to taille_du_tableau do
    begin
        writeln('tab[',i,']=',tab[i]:0:5);
    end;

    readln;
end.
```



4.2 Les tableaux à plusieurs dimensions

Les tableaux étudiés à la section précédente sont des tableaux à une seule dimension car un seul indice suffit pour les utiliser.

Les tableaux à deux dimensions auront besoin de deux indices, ceux de N dimensions auront besoin de N indices.

4.2.1 Les matrices

Une matrice est un tableau à deux dimensions, c'est-à-dire un tableau constitué de lignes et de colonnes (une grille).

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \\ m_{51} & m_{52} & m_{53} & m_{54} \end{bmatrix}$$

FIGURE 4.2 – Tableau à deux dimensions

On les déclare

```
var matrice : array [1..m,1..n] of integer ;
```

où m est le nombre de lignes et n, le nombre de colonnes.

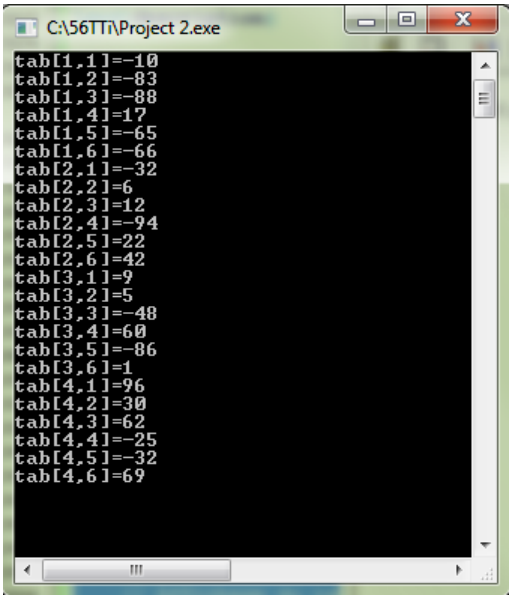
Accès aux éléments $matrice[3, 4]$ représente l'élément qui se trouve en troisième ligne et en quatrième colonne.

Exemple

Générer aléatoirement et afficher tous les éléments entiers d'un tableau de 4 lignes et 6 colonnes. Les éléments sont compris entre –100 et 100.

```
(*******************************************************
 * Générer aléatoirement et afficher tous les éléments *
 * entiers d'un tableau de 4 lignes et 6 colonnes.     *
 * Les éléments sont compris entre -100 et 100.        *
 *******************************************************)

 program tableau;
 var   lig,col : integer; // indices du tableau (ligne et colonne)
       tab : array[1..4,1..6] of integer; //tableau de 4*6 entiers
 begin
    // Remplissage du tableau
    randomize;
    for lig := 1 to 4 do
       for col := 1 to 6 do
          tab[lig,col] := -100 + random(201);
```

```
    // Affichage du tableau
    for lig := 1 to 4 do
        for col := 1 to 6 do
            writeln('tab[',lig,',',col,']=',tab[lig,col]);

    readln;
end.
```



4.2.2 Les tableaux d'une dimension supérieure à deux

On les déclare

```
var t : array [1..n1,1..n2,1..n3,...] of ... ;
```

et on accède aux éléments en écrivant :

```
t[i1,i2,i3,...]
```

4.3 Exercices

1. Multiplier tous les éléments d'un tableau par 4.
2. Calculer la somme de deux tableaux à une dimension (utiliser trois tableaux).
3. Calculer la somme et le produit des éléments d'un tableau.
4. Calculer le plus grand et le plus petit élément d'un tableau.
5. Inverser le premier et le dernier élément d'un tableau.

   Solution : on utilise une variable intermédiaire.

   ```
   Intermediaire := T[1] ;
   T[1] := T[n] ;
   T[n] := Intermediaire ;
   ```

6. Inverser tous les éléments d'un tableau en utilisant la technique suivante : le premier avec le deuxième, le deuxième avec le troisième, ... (permutation circulaire vers la gauche).
7. Inverser tous les éléments d'un tableau en utilisant la technique suivante : le dernier avec l'avant-dernier, l'avant-dernier avec l'avant-avant-dernier, ... (permutation circulaire vers la droite).
8. Permuter circulairement un tableau de k positions vers la droite ou vers la gauche. Indication : utiliser deux tableaux.
9. Étant donné un tableau de nombres tous différents, chercher si un nombre donné figure parmi les composants. Si oui, indiquer la valeur de l'indice correspondant (recherche séquentielle).
10. Même question, mais les composants du tableau peuvent être égales. Si le nombre à rechercher figure plusieurs fois parmi les éléments du tableau, on déterminera tous les indices correspondants. Ce n'est qu'après avoir déterminé tous les indices qu'on les écrira.
11. Jeu du « bonhomme pendu ». Le programme lit un mot proposé par un premier joueur. Il affiche ensuite le mot où toutes les lettres sauf la première et la dernière sont remplacées par un tiret. Un deuxième joueur propose des lettres. Chaque fois que la lettre se trouve dans le mot, le programme remplace autant de tirets que nécessaire par cette lettre et réaffiche le mot. Le second joueur a droit à un maximum de six essais infructueux.
12. Calculer le nombre moyen de jets de dés qu'il faut pour obtenir une première fois 6. Utiliser un tableau dont chaque élément sera, pour un essai donné, le nombre de jets qu'il a fallu exécuter pour avoir 6. Il restera à calculer la moyenne des données du tableau.
13. Calculer le nombre moyen de jets de dés qu'il faut pour obtenir une première fois 421 en lançant les trois dés en même temps. (Cf. exercice précédent).
14. Construire des carrés magiques d'ordre impair. Un carré magique d'ordre N est un tableau carré $(N \times N)$ dont les composants sont les N^2 premiers nombres entiers disposés de telle façon que les sommes des termes de chaque colonne, de chaque

ligne, ou de chaque diagonale principale soient égales.

Exemple : Carré magique d'ordre 3

8	1	6
3	5	7
4	9	2

Somme des termes d'une ligne = somme des termes d'une colonne = somme des termes d'une diagonale = 15

Lorsque N est impair, il existe un algorithme ; l'exposé sera fait sur un carré d'ordre 5, mais le raisonnement s'appliquerait à un carré d'ordre N.

(a) **Phase initiale**

Pour commencer, on place le chiffre 1 dans la case située au-dessus de celle du milieu du carré. On continue ensuite en plaçant les nombres 2, 3, 4, ... selon la diagonale ascendante de direction NORD-EST. Si I est l'indice de ligne et J l'indice de colonne, cette diagonale s'obtient en faisant $I = I-1$ et $J = J+1$:

23	6	19	2	15
10	18	**1**	14	22
17	5	13	21	9
4	12	25	8	16
11	24	7	20	3

(b) **Règles à observer lors de la progression dans le carré**

La loi de progression mentionnée ci-dessus entraînera tôt ou tard une sortie hors des limites du carré défini. La notation A(I,J) repérant l'élément courant du tableau A, on devra appliquer les règles suivantes :

- Si $I < 1$, on repart de la ligne du bas ($I = N$), tout en conservant le même indice J de colonne.
- Si $J > N$, on repart de la première colonne à gauche ($J = 1$), tout en conservant la valeur de l'indice I obtenu.
- Si $J < 1$, on repart de la dernière colonne ($J = N$), en conservant toujours l'indice ligne I.

Vérification de la case atteinte

- Il peut arriver que l'on atteigne une case contenant déjà un nombre rangé précédemment. On doit donc prévoir un moyen permettant de faire cette vérification : par exemple, lors de l'initialisation, on placera des zéros dans tous les éléments du carré, ce qui permettra ultérieurement de déduire si la case est libre ou non.
- Si le test fait apparaître une occupation de la case A(I,J), on choisit la direction NORD-OUEST à partir de la case courante occupée ; on reprendra la direction normale NORD-EST dès que le nombre courant est placé dans cette case "libre". La direction NORD-OUEST s'obtient en faisant $I = I - 1$ et $J = J - 1$, étant entendu que l'on devra vérifier que l'on ne sort pas du tableau (comme avant).

15. Calculer le produit de deux matrices.

4.4 Le tri d'un tableau

4.4.1 Le tri par sélection

Il existe le tri par sélection du minimum et le tri par sélection du maximum. Le tri qui suit est le tri par sélection du minimum.
On voudrait trier le tableau suivant : | 5 | -2 | 30 | 10 | 0 |

1. Étape 1
   - on recherche le minimum des éléments du tableau, c'est | -2 |
   - on échange le minimum avec le premier élément du tableau, on obtient : | -2 | 5 | 30 | 10 | 0 |
2. Étape 2
   - on recherche le minimum des éléments du tableau en commençant au 2e élément, c'est | 0 |
   - on échange le minimum avec le 2e élément du tableau, on obtient : | -2 | 0 | 30 | 10 | 5 |
3. Étape 3
   - on recherche le minimum des éléments du tableau en commençant au 3e élément, c'est | 5 |
   - on échange le minimum avec le 3e élément du tableau, on obtient : | -2 | 0 | 5 | 10 | 30 |
4. Étape 4
   - on recherche le minimum des éléments du tableau en commençant au 4e élément, c'est | 10 |
   - on échange le minimum avec le 4e élément du tableau, on obtient : | -2 | 0 | 5 | 10 | 30 |
   - il n'y a plus qu'un seul élément | 30 |, le tableau est donc trié.

Remarque : Il y une étape en moins que le nombre d'éléments du tableau.

4.4.2 Le tri « bulles »

On voudrait trier le tableau suivant : | 5 | -2 | 30 | 10 | 0 |

1. Étape 1
   - on compare les deux premiers éléments du tableau, on les échange s'ils ne sont pas dans l'ordre ; on obtient : | -2 | 5 | 30 | 10 | 0 |
   - on compare le 2e et le 3e élément du tableau, on les échange s'ils ne sont pas dans l'ordre ; on obtient : | -2 | 5 | 30 | 10 | 0 |

- on compare le 3e et le 4e élément du tableau, on les échange s'ils ne sont pas dans l'ordre ; on obtient : | -2 | 5 | 10 | 30 | 0 |
- on compare le 4e et le 5e élément du tableau, on les échange s'ils ne sont pas dans l'ordre ; on obtient : | -2 | 5 | 10 | 0 | 30 |

2. Étape 2
   - on compare les deux premiers éléments du tableau, on les échange s'ils ne sont pas dans l'ordre ; on obtient : | -2 | 5 | 10 | 0 | 30 |
   - on compare le 2e et le 3e élément du tableau, on les échange s'ils ne sont pas dans l'ordre ; on obtient : | -2 | 5 | 10 | 0 | 30 |
   - on compare le 3e et le 4e élément du tableau, on les échange s'ils ne sont pas dans l'ordre ; on obtient : | -2 | 5 | 0 | 10 | 30 |
   - on compare le 4e et le 5e élément du tableau, on les échange s'ils ne sont pas dans l'ordre ; on obtient : | -2 | 5 | 0 | 10 | 30 |
3. Étape 3
   - on compare les deux premiers éléments du tableau, on les échange s'ils ne sont pas dans l'ordre ; on obtient : | -2 | 5 | 0 | 10 | 30 |
   - on compare le 2e et le 3e élément du tableau, on les échange s'ils ne sont pas dans l'ordre ; on obtient : | -2 | 0 | 5 | 10 | 30 |
   - on compare le 3e et le 4e élément du tableau, on les échange s'ils ne sont pas dans l'ordre ; on obtient : | -2 | 0 | 5 | 10 | 30 |
   - on compare le 4e et le 5e élément du tableau, on les échange s'ils ne sont pas dans l'ordre ; on obtient : | -2 | 0 | 5 | 10 | 30 |
4. Étape 4
   - on compare les deux premiers éléments du tableau, on les échange s'ils ne sont pas dans l'ordre ; on obtient : | -2 | 0 | 5 | 10 | 30 |
   - on compare le 2e et le 3e élément du tableau, on les échange s'ils ne sont pas dans l'ordre ; on obtient : | -2 | 0 | 5 | 10 | 30 |
   - on compare le 3e et le 4e élément du tableau, on les échange s'ils ne sont pas dans l'ordre ; on obtient : | -2 | 0 | 5 | 10 | 30 |
   - on compare le 4e et le 5e élément du tableau, on les échange s'ils ne sont pas dans l'ordre ; on obtient : | -2 | 0 | 5 | 10 | 30 |

Remarque : pour cette étape, *aucun échange n'a eu lieu* : arrêt, le tableau est trié !

4.4.3 Le tri par insertion

On voudrait trier le tableau suivant : | 5 | -2 | 30 | 10 | 0 |

Étape 1
- on commence à partir du 2e élément du tableau | -2 |
- on insère cet élément à sa place dans la partie gauche du tableau en décalant les éléments nécessaires vers la droite | -2 | 5 | 30 | 10 | 0 |

Étape 2
- on considère l'élément suivant dans le tableau, c'est le 3e et il vaut | 30 |
- on insère cet élément à sa place dans la partie gauche du tableau en décalant les éléments nécessaires vers la droite (ici, 30 est déjà à sa place) | -2 | 5 | 30 | 10 | 0 |

Étape 3
- on considère l'élément suivant dans le tableau, c'est le 4e et il vaut | 10 |
- on insère cet élément à sa place dans la partie gauche du tableau en décalant les éléments nécessaires vers la droite | -2 | 5 | 10 | 30 | 0 |

Étape 4
- on considère l'élément suivant dans le tableau, c'est le 5e et il vaut | 0 |
- on insère cet élément à sa place dans la partie gauche du tableau en décalant les éléments nécessaires vers la droite | -2 | 0 | 5 | 10 | 30 |

Remarque Il y une étape en moins que le nombre d'éléments du tableau.

Indication Voici comment placer le ie élément dans la partie gauche

```
insere := T[i];
// Trouver la place d'insertion (ce sera j)
j := 1;
while T[i] > T[j] do
    j:=j+1;
// Décalage jusqu'à la position j
for k:=i-1 downto j do
    T[k+1] := T[k];
// Placer le ie élément dans le "trou"
T[j] := insere;
```

ou

```
// Mettre le ie élément au bon endroit dans la partie gauche du tableau
insere := T[i];
j := i-1;
while (j >= 1) and (T[j] > insere) do
begin
    T[j+1] := T[j];
    j := j-1;
end;
T[j+1] := insere;
```

4.4.4 Le tri à peigne

Le *comb sort* ou *tri à peigne* ou *tri de Dobosiewicz* est un algorithme de tri assez simple initialement inventé par Wodzimierz Dobosiewicz en 1980. Dans le tri à bulles ascendant classique, chacun des éléments de la liste à trier est comparé avec son voisin immédiat, et interverti s'il est plus grand que l'élément suivant.

L'idée de base du comb sort est de comparer un élément avec un autre élément plus lointain, espacé de celui-ci d'un certain intervalle. Au départ, cet intervalle est relativement grand, puis on le réduit progressivement à chaque passe de l'algorithme jusqu'à ce qui ne soit plus que de 1 en fin de traitement. Le tri à bulles peut être considéré comme une variante du comb sort dans lequel l'intervalle est toujours de 1.

La valeur d'origine de l'intervalle est généralement la taille de la liste à trier divisée par le facteur de réduction. Des essais empiriques ont montré que la valeur idéale de ce facteur de réduction est voisine de $1,3$.

Une démonstration animée du fonctionnement de ce tri se trouve à l'adresse
http ://lwh.free.fr/pages/algo/tri/tri_peigne.html

Autres tris D'autres tris seront étudiés en sixième.

4.5 La recherche dans un tableau

4.5.1 La recherche séquentielle

On appelle clé l'élément à rechercher. Le principe de la recherche séquentielle est de rechercher la clé en partant du premier élément du tableau jusqu'au dernier.

La recherche s'arrête dès que la clé est trouvée ou si le dernier élément du tableau est dépassé (dans ce dernier cas, la clé n'est pas dans le tableau).

Si la clé se trouve plusieurs fois dans le tableau, on peut demander à la recherche de

- s'arrêter à la première occurrence de la clé ;
- rechercher la dernière occurrence de la clé ;
- afficher tous les endroits où la clé se trouve.

4.5.2 La recherche dichotomique

La recherche dichotomique doit s'exécuter sur un tableau **trié**. Deux exemples illustreront son principe de fonctionnement.

Exemple 1 :

Considérons

- le tableau d'entiers suivant et ses indices :

1	3	4	6	9	11	12	14	17	20	24	40
1	2	3	4	5	6	7	8	9	10	11	12

- la clé $\boxed{9}$

1. On prend l'élément au milieu du tableau, ici $\boxed{11}$.
   On compare 11 à l'élément à rechercher (la clé 9) : comme $9 < 11$, on peut limiter la recherche à la partie *gauche* du tableau :

1	3	4	6	9
1	2	3	4	5

2. On prend l'élément au milieu (ici $\boxed{4}$) de ce tableau (qui est le tableau de départ où on ne s'intéresse plus qu'à certains indices).
   On compare 4 à la clé : comme $9 > 4$, on peut limiter la recherche à la partie *droite* du tableau :

6	9
4	5

3. On prend l'élément au milieu (ici $\boxed{6}$) de ce tableau.
   On compare 6 à la clé : comme $9 > 6$, on peut limiter la recherche à la partie *droite* du tableau :

9
5

4. On prend l'élément au milieu (ici $\boxed{9}$) de ce tableau.
   On compare 9 à la clé : comme $9 = 9$, on a trouvé la clé à la place 5.

Exemple 2 :

Considérons

- le tableau d'entiers suivant et ses indices (le même tableau que précédemment) :

1	3	4	6	9	11	12	14	17	20	24	40
1	2	3	4	5	6	7	8	9	10	11	12

- la clé $\boxed{22}$

1. On prend l'élément au milieu du tableau, ici $\boxed{11}$.
   On compare 11 à l'élément à rechercher (la clé 22) : comme $22 > 11$, on peut limiter la recherche à la partie *droite* du tableau :

12	14	17	20	24	40
7	8	9	10	11	12

2. On prend l'élément au milieu de ce tableau (ici $\boxed{17}$).
   On compare 17 à la clé : comme $22 > 17$, on peut limiter la recherche à la partie *droite* du tableau :

20	24	40
10	11	12

3. On prend l'élément au milieu (ici $\boxed{24}$) de ce tableau.
   On compare 24 à la clé : comme $22 < 24$, on peut limiter la recherche à la partie *gauche* du tableau :

20
10

4. On prend l'élément au milieu (ici $\boxed{20}$) de ce tableau.
   On compare 20 à la clé : comme $22 > 20$, on peut limiter la recherche à la partie *droite* du tableau **qui n'existe plus** ; on arrête en indiquant que la clé ne se trouve pas dans le tableau.

Exercice 1 combien d'étapes cette recherche comporte-t-elle au maximum si le tableau contient n éléments (essayer avec 100, 1000, 10000, puis n) ?

Exercice 2 programmer cette recherche.

Quelques indications

- utiliser trois **indices** qu'on va appeler $gauche$, $droite$ et $milieu$ où
  - $gauche$ représentera l'indice gauche du tableau qui reste à traiter ;
  - $droite$ représentera l'indice droite du tableau qui reste à traiter ;
  - $milieu$ représentera l'indice du milieu du tableau qui reste à traiter ;

  au départ, $gauche$ vaut 1, $droite$ vaut n (n est le nombre d'éléments du tableau) ;
- le calcul de milieu est simplement $\frac{gauche+droite}{2}$ (division entière) ;
- on se déplace à droite en écrivant $gauche \leftarrow milieu + 1$;
- on se déplace à gauche en écrivant $droite \leftarrow milieu - 1$;
- la recherche s'arrête si la clé est trouvée ou si $gauche > droite$

4.5.3 La fusion de deux tableaux triés

CHAPITRE 5

TYPES DE DONNÉES ADDITIONNELS

Quatre nouveaux types de données vont être étudiés dans cette partie : le type *énumération*, le type *intervalle*, le type *enregistrement* [1] et le type *ensemble* (le type fichier sera étudié dans un chapitre à part).

5.1 Le type énumération

Les types de base en Pascal sont les **types scalaires**. Les types scalaires sont composés d'éléments appelés constantes pour lesquelles il existe une relation d'ordre (une constante est plus petite, égale ou plus grande qu'une autre). Le nombre total de constantes dans le type scalaire est appelé la cardinalité de ce type. Le langage Pascal fournit quatre types scalaires : Integer, Real, Boolean et Char. On peut définir soi-même des types scalaires (une énumération). Voici un exemple :

```
type jourSemaine = (lun,mar,mer,jeu,ven,sam,dim);
   direction = (nord,est,sud,ouest);
   couleur = (pique,coeur,carreau,trefle);     // Couleurs des cartes
   etatCivil = (marie,veuf,celibataire);
var jour : jourSemaine;
   versOu : direction;
   carte : couleur;
   etat : etatCivil;
```

"jour" ne peut prendre que sept valeurs possibles (lun, mar, ..., dim), "versOu" et "carte" ne peuvent en prendre que quatre.

Écriture simplifiée l'extrait de programme ci-dessus peut également s'écrire sans définir explicitement des nouveaux types de données :

```
var jour : (lun,mar,mer,jeu,ven,sam,dim);
   versOu : (nord,est,sud,ouest);
   carte : (pique,coeur,carreau,trefle);
   etat : (marie,veuf,celibataire);
```

1. Des quatre types, le type *enregistrement* est le plus important !

On peut utiliser nos nouveaux types et variables de la manière suivante :

```
jour := mar;
if (etat = marie) or (etat = veuf) then
   ···;
for jour := lun to ven do ···;
for carte := trefle downto pique do ···;
if versOu <= sud then ···;
   ⋮
```

Remarque l'utilisation de constantes permet également de faciliter la lecture d'un programme :

```
const LUN = 1;
   MAR = 2;
   MER = 3;
   ⋮
```

Cette méthode est peut-être moins commode que l'utilisation d'un type énumération.

5.2 Le type intervalle

Il arrive fréquemment que quand on utilise un type scalaire, on ai besoin que d'une partie des valeurs possibles de ce type. Un type intervalle est un type composé d'une partie des valeurs possibles d'un type (sauf réel). On utilise un type **intervalle** en écrivant

```
type nom_type = limiteInferieure .. limiteSuperieure;
```

Quelques exemples :

```
1 type resultat = 0 .. 100;
2    alphabet = 'A' .. 'Z';
3    chiffre = '0' .. '9';    // Des caractères
4    jourOuvrable = lun .. ven; // du lundi au vendredi
5 var note : resultat;
6    lettre : alphabet;
7    jour : jourOuvrable;
```

La ligne 4 ne provoque pas d'erreur si "lun" et "ven" ont été définis avant. Par exemple,

```
type jourSemaine = (lun,mar,mer,jeu,ven,sam,dim);
   jourOuvrable = lun .. ven; // Aucun problème
```

Écriture simplifiée on peut aussi écrire :

```
1 var note : 0 .. 100;
2    lettre : 'A' .. 'Z';
3    jour : lun .. ven;
```

Avantages du type intervalle

- la ligne 1 est plus parlante (précise) que

```
var note : integer;
```

- une instruction du style

```
note := 101;
```

provoquera immédiatement un avertissement à la compilation
Warning : range check error while evaluating constants
tandis que

```
note := N;
```

provoquera une erreur à l'exécution (le programme s'arrête) si N n'est pas dans l'intervalle attendu ($[0, 100]$) et que la directive *{$RANGECHECKS ON}* est utilisée.

5.3 Le type enregistrement

5.3.1 Définition

Contrairement aux tableaux qui ne permettent pas de travailler avec des éléments de types différents, les **enregistrements** permettent au programme de définir des structures de variables regroupant des éléments de *types différents*.

Les composants d'un enregistrement sont appelés les **champs** de l'enregistrement. La définition d'un type enregistrement consiste à spécifier les différents champs, chacun d'eux étant identifié par son type et par un identificateur.

La déclaration d'un type enregistrement est la suivant :

```
type  nom_de_type = record
          champ_1 : type de donnée;
          champ_2 : type de donnée;
              ...
          champ_n : type de donnée;
      end;
```

Par exemple :

```
type  TEleve = record
          nom : String[20];
          prenom : String[20];
          classe : String[4];
          sexe : char;
          dateNaissance : TDate;   // ou TDateTime
          note : integer;
      end;
```

Une fois que la structure de l'enregistrement a été définie, on peut déclarer des variables du type voulu. Si on écrit :

```
var eleve1, eleve2 : TEleve;   // Deux variables "simples"
    tableauEleves : array[1..1000] of TEleve; // Tableau d'élèves
    fichierEleves : file of TEleve;   // Fichiers d'élèves
```

on définit quatre variables *eleve1*, *eleve2*, *tableauEleves* et *fichierEleves*.

5.3.2 Accès aux champs

On accède aux différents champs d'un variable de type enregistrement en utilisant un point entre le nom de la variable et le nom du champ :

nom_de_variable.champ

Par exemple,

```
eleve1.nom := 'DURANT';
write('Donner un prenom : ');
readln(eleve1.prenom);
eleve2.note := eleve2.note + 10;

for i:=1 to 10 do
   tableauEleves[i].sexe := 'F';
```

5.3.3 Exemple

Un point peut se représenter par un enregistrement formé de deux champs qui sont ses coordonnées. Écrire un programme qui calcule la distance entre deux points.

```
program PPoints;
type TPoints = record
          abscisse : real;
          ordonnee : real;
       end;
var pt1, pt2 : TPoints;
   distance : real;
begin
   // pt1 = (1/2, 3.0)
   pt1.abscisse := 1/2;
   pt1.ordonnee := 3.0;
   // pt2 = (4.0, 3/4)
   pt2.abscisse := 4.0;
   pt2.ordonnee := 3/4;

   // la distance entre (x1,y1) et (x2,y2) vaut sqrt((x2-x1)^2 + (y2-y1)^2)
   distance := sqrt(
      (pt2.abscisse-pt1.abscisse)*(pt2.abscisse-pt1.abscisse) +
      (pt2.ordonnee-pt1.ordonnee)*(pt2.ordonnee-pt1.ordonnee));

   writeln('Distance = ',distance:0:4);

   readln;
end.
```

Remarque on reviendra sur le type *enregistrement* lors de l'étude des fichiers.

5.4 Le type ensemble

5.4.1 Introduction

En programmation, les ensembles sont mis en œuvre avec la même signification qu'ils possèdent en mathématique. On y retrouve d'ailleurs les mêmes opérations. Pour mieux comprendre l'utilité du type ensemble, considérons le problème suivant.
Comment savoir si un caractère (*Key*) est une voyelle majuscule ?
Une structure sélective permet de résoudre facilement ce problème :

```
if (Key='A') or (Key='E') or (Key='I') or (Key='0') or
   (Key='U') or (Key='Y') then
      write ('C''est une voyelle');
```

Ce genre de problème, consistant à déterminer si le contenu d'une variable appartient à un ensemble de valeurs données, se rencontre souvent en programmation. Voici comment il est possible de résoudre ce problème à l'aide d'un ensemble :

```
if Key in ['A','E','I','0','U','Y'] then
   write ('C''est une voyelle');
```

Ici, l'ensemble des voyelles est spécifié de manière explicite par une liste de constantes placée entre crochets, appelée constructeur d'ensemble. Cette forme d'écriture abrégée est bien plus élégante, plus claire et plus concise qu'une expression booléenne complexe. L'ensemble des voyelles peut, dans cet exemple, être considéré comme une constante de type ensemble. Un ensemble contient toujours des éléments du même type de base, scalaire (prédéfini ou énuméré) ou intervalle, à l'exception du type réel. Le type ensemble se déclare de la manière suivante :

```
type type_ensemble = set of type_de_base;
```

Une variable de type type_ensemble peut contenir des éléments de type type_de_base, qui doit être un type scalaire ou intervalle, à l'exception du type réel. Le *nombre maximum d'éléments* d'un ensemble est limité à *256*. De plus, deux ensembles sont égaux s'ils contiennent les mêmes éléments, indépendamment de leur l'ordre. La définition d'une variable de type ensemble comporte la spécification du type de base des éléments de cet ensemble. Seule une affectation permet de placer des valeurs dans une variable de ce type. L'erreur consistant à croire que la déclaration d'une telle variable suffit à lui attribuer des éléments se rencontre fréquemment [2].

L'affectation d'éléments à un ensemble s'effectue à l'aide d'un constructeur d'ensemble qui n'est autre qu'une liste d'éléments du type de base, délimitée par des crochets. La notation d'intervalle est possible lorsque plusieurs éléments sont consécutifs. Le fragment de programme qui suit montre quelques exemples de déclaration et d'utilisation de variables de type ensemble :

2. Cf. page suivante

```
 1  type alphabet = set of 'A'..'Z';
 2     couleurs = (rouge, vert, bleu, jaune, blanc);
 3     teinte   = set of couleurs;
 4     bizarre  = set of 0..255;
 5
 6  var lettre,voyelle : alphabet;
 7     c        : char;
 8     lumiere  : teinte;
 9     clair    : set of jaune..blanc;
10     magique  : bizarre;
11
12  begin
13     lettre  := ['A'..'Z'];
14     voyelle := ['A','E','I','0','U','Y'];
15     lumiere := [blanc];
16     clair   := [jaune, blanc] ;
17     magique := [0..8,12,19,22..38,99];
18     if c in voyelle then
19        write('Voyelle');
20  end;
```

- les lignes $1 \longrightarrow 4$ définissent quatre nouveaux types de données ;
- les lignes $6 \longrightarrow 10$ déclarent des variables dont le contenu est, à ce moment, indéterminé ;
- les lignes $13 \longrightarrow 17$ donnent des valeurs aux variables.

Le langage Pascal dispose d'opérateurs agissant sur les ensembles, de manière analogue aux opérateurs de la théorie des ensembles en mathématique. On trouve, d'un côté, les opérateurs ensemblistes proprement dits (union, intersection et différence) et, de l'autre, des opérateurs relationnels (appartenance, égalité, altérité, inclusion). Comme pour les nombres, ces opérateurs ont un degré de priorité et, en cas de même degré de priorité, l'évaluation des expressions se fait de gauche à droite.

Voyons comment s'expriment ces opérateurs en Pascal. Les variables utilisées sont déclarées comme suit :

```
var nombres,resultat,a,b : set of 0..10;
```

5.4.2 Manipulation des ensembles

Supposons que

```
nombres := [1..10];
a := [1,2,4];
b := [3,5] ;
```

Intersection

$$A \cap B = \{x \mid x \in A \, et \, x \in B\}$$

En Pascal, l'intersection est utilisée grâce à "*"

```
resultat := nombres*[2,4..6]; // resultat aura la valeur [2,4,5,6]
if a * b = [] then    // [] représente l'ensemble vide.
   write ('a et b sont disjoints');
```

Union

$$A \cup B = \{x \mid x \in A \, ou \, x \in B\}$$

En Pascal, l'union est utilisée grâce à "+"

```
resultat:=a+b+[6,7,8,9]; // resultat aura la valeur [1,2,3,4,5,6,7,8,9]
```

Différence

$$A \setminus B = \{x \mid x \in A \, et \, x \notin B\}$$

En Pascal, la différence est utilisée grâce à "-"

```
resultat := nombres-[1,3..7]; // resultat aura la valeur [2,8,9]
```

Égalité

$A = B$ **si** A **et** B **contiennent les mêmes éléments**

En Pascal, l'égalité est utilisée grâce à "="

```
if nombres = [1..10] then
   write ('Ensembles égaux') ;
```

Altérité

$A \neq B$ **si** A **et** B **ne contiennent pas les mêmes éléments**

En Pascal, l'altérité est utilisée grâce à "<>"

```
if (nombres <> [1..10] ) then
   write ('Ensembles non égaux') ;
```

Inclusion

$A \subset B$ **si tous les éléments de** A **sont dans** B

En Pascal, l'inclusion est utilisée grâce à "<="

```
if nombres <= [0,2,4,6,8] then
   chaine := 'pair';
if (a <= b) and (a <> b) then
   chaine := 'a inclus mais non égal a b';
```

Appartenance

$a \in A$ **si** a **est un élément de** A

En Pascal, l'appartenance est utilisée grâce à "in"

```
if caractere in ['A','E','I','O','U','Y'] then
   write ('Voyelle');
```

Remarque

Il est important de ne pas confondre la notion d'ensemble avec la notion d'intervalle ou de type énuméré. Si, par exemple, on désire écrire :

```
caractere := ['A'..'L'] ;
```

caractere peut, par exemple, être déclaré comme :

```
var caractere : set of 'A'..'L';
```

et non comme :

```
var caractere : 'A'..'L';   // ici caractere n'est pas un ensemble
```

Test

Supposons que

```
A = [1,3,5,7,9]
B = [2,4,6,8,10]
C = [1,2,3,4,5]
D = [5]
E = [ ]
```

Quelle est la valeur de chacune des opérations suivantes ?

1. (A+B)-C
2. (A*C)=D
3. (A+E)*(B+D)
4. 7 IN (((A-B)-C)-D)
5. C <= (A+B)

5.4.3 Exemple

Calculer le nombre de voyelles, de consonnes et des autres caractères d'une chaîne (exercice 24 page 59).

```
program lettres;
var voyelles, consonnes : set of char;
    chaine : String;
    nbVoyelles, nbConsonnes, nbAutres : integer;
    i : integer;
begin
    voyelles := ['A','E','I','O','U','Y'];
    consonnes := ['B','C','D','F','G','H','J','K','L','M','N',
                  'P','Q','R','S','T','V','W','X','Z'];
    // Lire la chaine au clavier et la mettre en majuscules
    readln (chaine);
    chaine := upCase (chaine) ;

    // Calculs
    nbVoyelles := 0; nbConsonnes := 0; nbAutres := 0;
    for i:=1 to Length(chaine) do
    begin
        if chaine[i] IN voyelles then
            nbVoyelles := nbVoyelles + 1
        else
            if chaine[i] IN consonnes then
                nbConsonnes := nbConsonnes + 1
            else
                nbAutres := nbAutres + 1;
    end;

    // Afficher les résultats
    writeln (nbVoyelles, ' voyelle(s)');
    writeln (nbConsonnes, ' consonne(s)');
    writeln (nbAutres, ' autre(s)');

    readln;
end.
```

CHAPITRE 6

LES SOUS-PROGRAMMES

6.1 Décomposition d'un problème

En général, tout problème peut se décomposer en sous-problèmes plus simples à résoudre ; ces sous-problèmes pouvant être eux-mêmes décomposés, ... Cette décomposition s'arrête quand les sous-problèmes deviennent élémentaires à résoudre.

Nous allons illustrer cette décomposition grâce à deux exemples tirés de la vie courante.

Exemple 1

Décomposition de la tâche qui consiste à téléphoner à partir d'une cabine publique. Cette tâche sera décomposée en quatre sous-actions :

- obtenir le numéro désiré ;
- établir la communication avec son interlocuteur ;
- échanger un dialogue avec son interlocuteur ;
- raccrocher.

Ces tâches se décomposent elles-aussi. En effet, chacune d'elles soulève un obstacle. On obtient le numéro de la personne qu'on appelle en recherchant son nom dans son agenda ou dans l'annuaire ou en demandant l'information au centre de renseignements. Ensuite, il s'agira de mener à bien la deuxième tâche : établir la connexion avec son interlocuteur, ...

Cette décomposition peut se schématiser de la manière suivante :

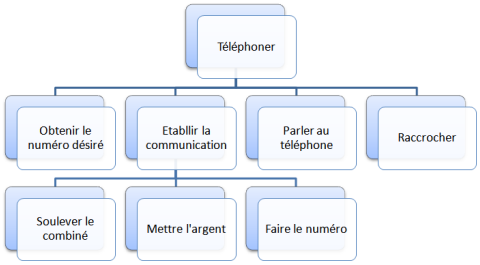


FIGURE 6.1 – Décomposition d'un problème (1)

Exemple 2

Décomposition de la tâche qui consiste à gérer un carnet d'adresses. On suppose que notre gestion se limite à trois opérations :

- ajouter un nom ;
- rechercher l'adresse ;
- éditer (modifier).

Bien entendu, un tel niveau de décomposition ne suffit pas pour pouvoir être exécuté sur une machine. En effet, un ordinateur ne peut effectuer directement l'action "ajouter une personne" ou "éditer un répertoire".
Il va donc falloir expliciter un peu le déroulement des opérations.

Ajouter une personne, c'est :

- acquérir les coordonnées ;
- chercher une place libre ;
- écrire dans le carnet.

Retrouver une personne, c'est :

- lire une identité ;
- rechercher dans le répertoire.

La structure du programme pourrait être la suivante :

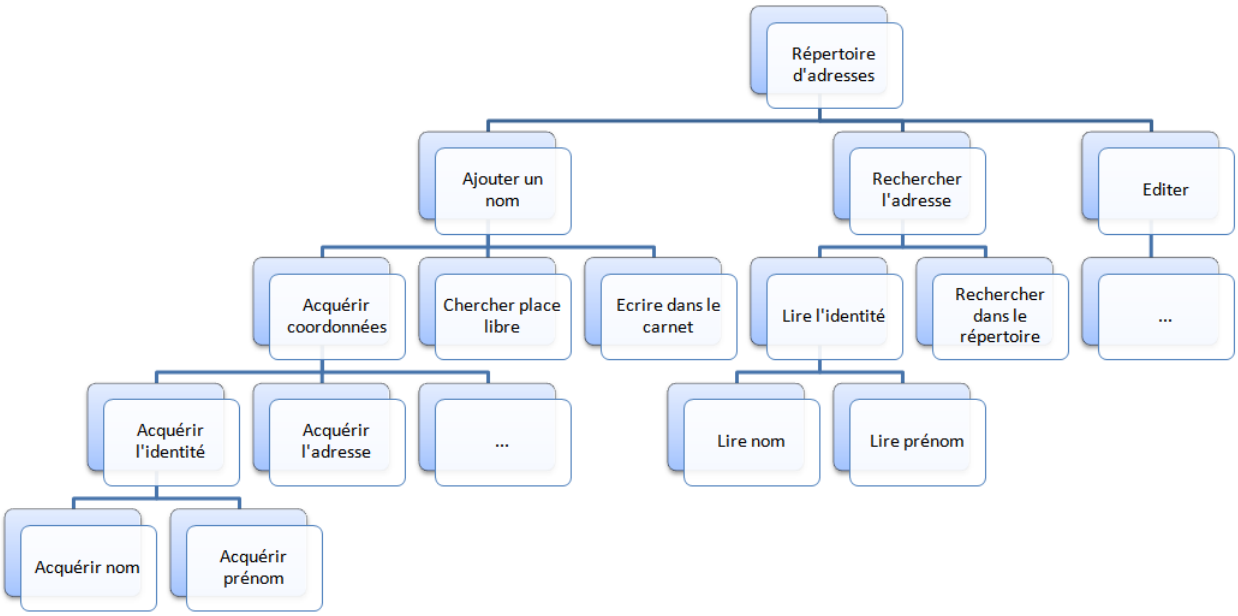


FIGURE 6.2 – Décomposition d'un problème (2)

La démarche employée dans ces deux exemples est une décomposition **descendante** aussi appelée par **raffinements successifs** [1] car elle consiste, une fois le but d'une tâche défini, à examiner les moyens nécessaires pour y parvenir et ainsi à décrire des opérations plus **élémentaires**.

1. En anglais : *top-down program design*

La notion de **sous-programme** est un outil informatique qui rend les programmes plus **modulaires**, c'est-à-dire les subdivise en entités plus simples, bien individualisées, et dont la fonction peut se comprendre et se déterminer aisément.

6.2 L'écriture de "gros" programmes

La partie la plus couteuse quant à l'utilisation des ordinateurs n'est pas le matériel lui-même mais le développement et la maintenance des programmes. Il faut donc des programmes clairs et facilement compréhensibles.

Conditions nécessaires pour obtenir des programmes lisibles

1. **Modularisation** Ne sont lisibles et clairs que les petits programmes. Pour illustrer cette affirmation, considérons le rôle que joue une variable à l'intérieur d'un programme : supposons que K soit une variable entière et qu'en lisant le programme, on rencontre une instruction qui commence par

   ```
   if K < 0 then ...
   ```

   Si le programme est suffisamment petit, le lecteur n'a probablement pas oublié la valeur de la variable K. Au pire, même s'il a perdu de vue cette valeur, les instructions qui ont assigné une valeur à K restent à la vue du lecteur qui peut les retrouver d'un seul coup d'œil. De plus, toutes les variables du programme (qui sont en nombre relativement réduit puisque le programme est petit) peuvent être "suivies" par le lecteur. En résumé, on dira qu'un programme est trop gros si les instructions qui intéressent le lecteur ne sont pas visibles d'un coup d'œil ou si elles ne sont pas faciles à trouver.

   Cet exemple montre la nécessité d'écrire de petits programmes (une page de listing devait être suffisante), c'est pourquoi il est **obligatoire que le programmeur découpe son programme en sous-programmes**. Cette technique est appelée la **découpe en modules** (ou modularisation).

   > La découpe modulaire doit être réfléchie : le programmeur doit **analyser** (penser) d'abord et **coder ensuite**.

2. **Documentation** Ce n'est jamais une perte de temps que de documenter ses programmes. Les **commentaires intelligents** sont utiles au concepteur (celui qui écrit le programme) et au lecteur (celui qui sera chargé de la maintenance).

   Au minimum, chaque sous-programme devrait contenir des commentaires expliquant :

   - les paramètres qui entrent dans le sous-programme ;
   - les paramètres qui en sortent ;
   - les actions (l'action) que fait le sous-programme.

   Par exemple,

```
function fact(n:integer):real;
(*
   En entrée, la fonction reçoit un entier n, elle ressort un réel.
   Elle calcule la factorielle de n où

                    fact(n) = n! = n × (n − 1) × (n − 2) × ··· × 3 × 2 × 1

*)
```

   Remarques - les commentaires ne devraient pas être écrits quand tout le travail est fini, mais bien au **début** ou pendant l'écriture du code ;
   - écrire des commentaires pour le seul plaisir d'en avoir est parfaitement inutile. Il faut que les commentaires apportent une information **pertinente**.

3. **Noms des identificateurs** Un programme (ou un sous-programme) sera d'autant plus lisible que les noms des identificateurs (qu'ils désignent des types, constantes, variables, fonctions, procédures, ...) auront été bien choisis. Ces identificateurs doivent "parler" d'eux-mêmes.

4. **Utilisation judicieuse des structures de contrôle** Comparer les extraits de programmes suivants :

```
somme := 0;
for i := 1 to 100 do
      somme := somme + i;
```

```
   somme := 0;
   i := 1;
ou : if i <= 100 then
   begin
      somme := somme + i;
      i := i+1;
      goto ou;
   end;
```

```
premier := true;
i := 2;
while (i <= nb - 1) and
      (premier = true) do
begin
   if nb mod i = 0 then
      premier := false;
   i := i + 1;
end;
```

```
premier := true;
for i := 2 to nb - 1 do
begin
   if nb mod i = 0 then
   begin
      premier := false;
      break;
   end;
end;
```

Pour le second exemple, cf. page 56

Un troisième exemple est l'utilisation d'un choix multiple plutôt qu'une série de structures alternatives. Ce cas a été étudié en page 42.

5. **Relecture ultérieure du programme** Une méthode pour tester la lisibilité d'un programme pourrait être celle consistant à relire celui-ci quelque temps après sa conception. Si le programmeur ne peut comprendre son propre programme immédiatement, c'est que le code n'est pas suffisamment clair et il doit être modifié.
6. **Utilisation d'une présentation adéquate** Le programme qui suit est tout à fait "compréhensible" par une machine mais pour une personne ...

```
program DiviseursNombre; var nombre, diviseur : integer;  begin
Write ('Quel est le nombre ? '); Readln(nombre); diviseur
:=1; while diviseur<=nombre do begin if nombre mod diviseur
 = 0 then begin Writeln(diviseur,' est un diviseur de ',nombre);
 end; diviseur:=diviseur+1; end; Readln; end.
```

Afin de clarifier encore davantage la clarté d'un programme, des commentaires dans le genre de ceux qui suivent peuvent s'avérer très utiles :

```
case ... of
   .
   .
   .
end;   (* Fin du Case Of *)
```

```
while ... do
begin
   .
   .
   .
   if ... then
   begin
      .
      .
      .
   end;   (* Fin du If*)
   .
   .
   .
end;   (* Fin de la boucle While *)
```

Attention La présentation peut parfois s'avérer trompeuse :

```
if A > 0 then
   if B > 0 then
      B := B+1
else
   A := A+1;
```

```
somme := 0;
i := 1;
while i < 100 do
   somme := somme + i;
   i := i+1;
writeln(somme);
```

6.3 Les procédures et fonctions

A la section précédente, on a vu la nécessité d'écrire des sous-programmes pour permettre la décomposition d'un programme en modules [2] plus simples.

Un autre avantage des sous-programmes est leur **utilisation ultérieure**. Si on est amené dans certains programmes à utiliser à plusieurs reprises une **même séquence d'instructions**, on souhaite alors ne pas devoir recopier systématiquement ces instructions. Ainsi,

Les sous-programmes permettent d'éviter de réécrire plusieurs fois la même chose.

Les **procédures** et les **fonctions** sont les deux types de sous-programmes qui existent dans le langage Pascal.

Leur utilisation présente, au début, quelques difficultés. Il est important de maîtriser ces difficultés si on veut progresser dans l'élaboration de bons programmes.

6.3.1 Les fonctions

1. **Définition**
   Une **fonction** est un sous-programme qui fournit **un résultat**, elle calcule une valeur.
   Par exemples,
   - calculer la racine carrée d'un nombre réel ;
   - transformer un nombre d'heures, minutes, secondes en secondes ;
   - calculer la somme des N premiers nombres entiers ;
   - calculer le PGCD de deux entiers ;
   - tester si un nombre est premier ;
   - calculer a^n où a est un nombre réel non nul et n est un nombre entier ;
   - calculer le nombre de voyelles d'une chaîne de caractères ;
   - mettre une chaîne en majuscules ;
   - extraire une partie de chaîne ;
   - calculer le nombre de mots d'un texte ;
   - calculer l'élément maximum d'un tableau de réels ;
   - calculer le nombre de fois que se trouve un entier dans un tableau d'entiers ;
   - ...

   Pour tous les énoncés qui précèdent, | **un seul résultat est calculé**. |

2. Les modules peuvent être programmés par des personnes différentes

L'écriture d'une fonction impose qu'on se pose deux questions :

(a) Quel type de résultat la fonction calcule-t-elle ?

(b) Que doit connaître la fonction pour s'exécuter ?

Reprenons les exemples ci-dessus :

Enoncé	**Résultat**	**Que doit connaître la fonction pour s'exécuter ?**
Calculer la racine carrée d'un nombre réel	Réel	Un nombre réel
Transformer un nombre d'heures, minutes, secondes en secondes	Entier	Trois nombres entiers
Calculer la somme des N premiers nombres entiers	Entier	Un nombre entier
Calculer le PGCD de deux entiers	Entier	Deux nombres entiers
Calculer a^n où a est un nombre réel non nul et n est un nombre entier	Réel	Un nombre réel et un nombre entier
Tester si un nombre est premier	Booléen	Un nombre entier
Calculer le nombre de voyelles d'une chaîne de caractères	Entier	Une chaîne
Mettre une chaîne en majuscules	Chaîne	Une chaîne
Extraire une partie de chaîne	Chaîne	Une chaîne et deux entiers
Calculer le nombre de mots d'un texte	Entier	Un fichier
Calculer l'élément maximum d'un tableau de réels	Réel	Un tableau de réels
Calculer le nombre de fois que se trouve un entier dans un tableau d'entiers	Entier	Un tableau d'entiers et un entier

FIGURE 6.3 – Écriture d'entêtes de fonctions

La troisième colonne indique les **paramètres** de la fonction.

2. **Déclaration d'une fonction**
   Une fonction se déclare de la manière suivante :

   function nom_de_la_fonction (liste_des_paramètres) : type_de_résultat ;

   Reprenons nos exemples,

   - calculer la racine carrée d'un nombre réel :
     ```
     function racine(x:real):real;
     ```
   - transformer un nombre d'heures, minutes, secondes en secondes :
     ```
     function HMS (heures,minutes,secondes:integer) : integer;
     ```
   - calculer la somme des N premiers nombres entiers :
     ```
     function sommeN (n : integer) : integer:
     ```
   - calculer le PGCD de deux entiers :
     ```
     function PGCD(n1,n2 : integer) : integer:
     ```
   - calculer a^n où a est un nombre réel non nul et n est un nombre entier :
     ```
     function puissance (a:real; n:integer) : real;
     ```
   - tester si un nombre est premier :
     ```
     function estPremier (n : integer) : boolean;
     ```
   - calculer le nombre de voyelles d'une chaîne de caractères :
     ```
     function voyelles (s : string) : integer;
     ```
   - mettre une chaîne en majuscules :
     ```
     function majuscules (s : string) :string;
     ```
   - extraire une partie de chaîne :
     ```
     function extract (s : string; ou, combien : integer) : string:
     ```
   - calculer le nombre de mots d'un texte :
     ```
     function nbMots (f : Text) : integer;
     ```
   - calculer l'élément maximum d'un tableau de réels :
     ```
     function maxTab (tab : tableauReels; n : integer) : real;
     ```
     où *tableauReels* est défini comme
     ```
     type tableauReels = array[1..1000] of real;
     ```
     et n est le nombre d'éléments réellement utilisés du tableau
   - calculer le nombre de fois que se trouve un entier dans un tableau d'entiers :
     ```
     function nbFois (tab : tableauEntiers;n :integer) : integer;
     ```
     où *tableauEntiers* est défini comme
     ```
     type tableauEntiers = array[1..10000] of integer;
     ```
     et n est le nombre d'éléments réellement utilisés du tableau

3. **Appel d'une fonction**
   L'appel d'une fonction est son utilisation (on se sert de la fonction). Il suffit d'écrire le nom de la fonction et de respecter le nombre et le type des paramètres. Par exemple,

   - appels de la fonction *racine*

     ```
     y := racine(2.0); // y ≈ 1.414
     z := racine (8);  // z ≈ 2.828
     writeln (racine (y+z):0:5);     // affiche 2,05977
     ```

     où x,y et z sont des réels

   - appels de la fonction HMS

     ```
     nbSecondes := HMS (5,28,54);  // nbSecondes vaut 19734
     nbSecondes := nbSecondes + HMS(h,m,s);
     ```

     où h,m,s,$nbSecondes$ sont des entiers

   - calculer la somme des N premiers nombres entiers :

     ```
     somme := sommeN (100);  // somme vaut 5050
     writeln (sommeN(sommeN(5)));  // affiche 120
     ```

   - calculer le PGCD de deux entiers :

     ```
     resultat := PGCD(64,40);     // resultat vaut 8
     ```

   - ...

4. **Programme complet**

```
1  program exempleFonction;
2
3  (************ Fonction qui calcule 1+2+3+ ... +N ************)
4  function sommeN(n:integer):integer;
5  var som : integer; // Variable de travail
6      i   : integer; // Indice de boucle
7  begin
8     som := 0;
9     for i := 1 to n do
10       som := som + i;
11    Result := som; // ou sommeN := som;
12 end;
13
14 (******** Fonction qui calcule le nombre de voyelles *********)
15 function nbVoyelles(s:string) : integer;
16 const VOYELLES = 'aeiouyAEIOUY'; // Constante qui contient toutes les voyelles
17 var cptVoy : integer;  // Compteur de voyelles
18     i      : integer;  // Indice de boucle
19 begin
20    cptVoy := 0;
21    for i := 1 to Length(s) do
22       if Pos(s[i], VOYELLES) > 0 then
23          inc(cptVoy);    // inc(v) signifie "augmenter v de 1"
24    Result := cptVoy;  // ou nbVoyelles := cptVoy;
25 end;
26
27 (******************* Programme principal ********************)
28 var nombre : integer;
29     chaine : string;
30     cptVoy : integer;
31 begin
32    write ('Veuillez entrer un entier : ');
33    readln(nombre);
34    writeln('1+2+3+...+',nombre, ' vaut ', sommeN(nombre));
35
36    write ('Veuillez entrer une chaine : ');
37    readln(chaine);
38    cptVoy := nbVoyelles(chaine);
39    writeln(chaine,' contient ',cptVoy,' voyelle(s)');
40
41    readln;
42 end.
```

Commentaires

(a) La ligne 4 est la **déclaration** de la fonction. Elle comprend le nom de la fonction (*sommeN*), la liste des **paramètres formels** (*n :integer*) et le **type de résultat** (*integer*).

La ligne 15 est la déclaration de la fonction. Elle comprend le nom (*nbVoyelles*), la liste des paramètres formels (*s :string*) et le type de résultat (*integer*).

(b) Les lignes 5 et 6 sont les variables **locales** de la fonction. Ces variables ne sont connues que dans la fonction où elles sont déclarées.

Les lignes 16, 17 et 18 sont les variables et constante locales de la fonction. Ces variables et constante ne sont utilisables que dans la fonction où elles sont déclarées.

(c) Les lignes $5 \rightarrow 12$ forment le **corps** de la fonction.
Les lignes $16 \rightarrow 25$ forment le corps de la fonction.

(d) La ligne 11 est le **résultat** calculé par la fonction.
La ligne 24 est le résultat calculé par la fonction.

(e) Les lignes $28 \rightarrow 42$ forment le **programme principal** (c'est ce qui s'exécute).

i. Les lignes 28, 29 et 30 sont les variables du programme principal [3].

ii. La ligne 34 est l'**appel** (exécution) de la fonction et *nombre* est le **paramètre réel**. Les paramètres formels et réels n'ont pas obligatoirement le même nom mais doivent être de même type.
La ligne 38 est l'appel de la fonction et *chaine* est le paramètre réel. Les paramètres formels et réels n'ont pas obligatoirement le même nom mais doivent être de même type.

iii. Au moment de l'appel, le paramètre réel est **copié** dans le paramètre formel puis la fonction s'exécute.

6.3.2 Les procédures

1. **Définitions**
Les fonctions sont parfois d'un usage trop restrictif parce qu'elles ont pour but de calculer une valeur unique. Les **procédures** lèvent cette restriction.
Exemples :

(a) on utilisera une procédure pour transformer un nombre de secondes en heures, minutes, secondes (4000 secondes = 1H6'40"). Il y a **trois valeurs à calculer**.

(b) on utilisera une procédure pour dessiner un rectangle dont on connaît la largeur et la hauteur [a]. Il n'y a **rien à calculer**.

a. on suppose qu'on dessine le rectangle à l'endroit où se trouve le curseur

```
Veuillez entrer une largeur : 20
Veuillez entrer une hauteur : 6
××××××××××××××××××××
×                  ×
×                  ×
×                  ×
×                  ×
××××××××××××××××××××
```

Comme les fonctions, les procédures auront un **nom**, des **paramètres formels** et un **corps** comprenant une partie **déclarative** et une **séquence d'instructions**. Contrairement aux fonctions, les procédures ne retournent aucune valeur.

Une procédure se déclare de la manière suivante :

procedure nom_de_la_procedure (liste_des_paramètres) ;

3. Si les variables avaient été déclarées au début (ligne 2), elles devenaient des **variables globales**

On distingue deux catégories de paramètres formels, selon que leur valeur doit être connue au début de l'exécution de la procédure ou calculée dans la procédure. On appelle **paramètres d'entrée** ceux dont la valeur doit être connue pour que les instructions de la procédure puissent être exécutées, mais qui ne doivent pas subir de modification. Les paramètres pour lesquels la procédure détermine une valeur sont les **paramètres de sortie**.

(a) pour le premier exemple, il y a un paramètre d'entrée (le nombre de secondes) et trois paramètres de sortie (le nombre d'heures, de minutes, de secondes). On la déclare :

```
procedure HMS(secondes:integer; var h,m,s:integer);
```

les paramètres de sortie sont précédés du mot *var*.

(b) pour le deuxième exemple, il n'y a que des paramètres d'entrée (coordonnées de deux points). On la déclare :

```
procedure dessine(largeur,hauteur:integer);
```

2. **Programme complet**

```
1  program exempleProcedure;
2
3  (************* Procedure qui transforme en HMS *************)
4  procedure HMS(secondes:integer; var h,m,s:integer);
5  var reste : integer; // Variable de travail
6  begin
7     h := secondes div 3600;
8     reste := secondes mod 3600;
9     m := reste div 60;
10    s := reste mod 60;
11 end;
12
13 (************** Procédure qui dessine **************)
14 procedure dessine(largeur,hauteur:integer);
15 var   ligneEtoiles : string; (*****************************)
16       ligneBlancs  : string; (*                           *)
17       i : integer;
18 begin
19    ligneEtoiles := StringOfChar('*',largeur);
20    ligneBlancs  := '*' + StringOfChar(' ',largeur-2) + '*';
21
22    writeln(ligneEtoiles);
23    for i:=1 to hauteur-2 do
24       writeln(ligneBlancs);
25    writeln(ligneEtoiles);
26 end;
27
```

```
28  (****************** Programme principal ******************)
29  var largeur, hauteur : integer;
30      secondesDepart,heures,minutes,secondes : integer;
31  begin
32      write ('Veuillez entrer une largeur : ');
33      readln(largeur);
34      write ('Veuillez entrer une hauteur : ');
35      readln(hauteur);
36      dessine(largeur, hauteur);
37
38      write ('Veuillez entrer un nombre de secondes : ');
39      readln(secondesDepart);
40      HMS(secondesDepart,heures,minutes,secondes);
41      write(secondesDepart,' secondes = ',heures,'H',minutes,"",secondes,"""");
42
43      readln;
44  end.
```

Commentaires

(a) La ligne 4 est la **déclaration** de la procédure. Elle comprend le nom de la procédure (*HMS*) et la liste des **paramètres formels** (*secondes :integer; var h,m,s :integer*).
On écrit *var*[4] devant les paramètres de sortie (ceux qui sont calculés dans la procédure).
La ligne 14 est la déclaration de la procédure. Elle comprend le nom de la procédure (*dessine*) et la liste des paramètres formels (*largeur,hauteur :integer*).

(b) La ligne 5 est la variable **locale** de la procédure. Cette variable n'est connue que dans la procédure où elle est déclarée.
Les lignes 15, 16 et 17 sont les variables locales de la procédure.

(c) Les lignes $6 \rightarrow 11$ forment le **corps** de la procédure.
Les lignes $18 \rightarrow 26$ forment le corps de la procédure.

(d) Les lignes $31 \rightarrow 44$ forment le **programme principal** (c'est ce qui s'exécute).

i. Les lignes 29 et 30 sont les variables du programme principal[5].

ii. La ligne 36 est l'**appel** (exécution) de la procédure et *largeur,hauteur* sont les **paramètres réels**. Les paramètres formels et réels n'ont pas obligatoirement le même nom mais doivent être de même type.
La ligne 40 est l'appel de la fonction et *secondesDepart,heures,minutes,secondes* sont les paramètres réels. Les paramètres formels et réels n'ont pas obligatoirement le même nom mais doivent être de même type.

iii. Au moment de l'appel, les paramètres réels sont substitués aux paramètres formels puis la procédure est exécutée. Le point suivant explique plus en détail cette substitution qui est appelée le **passage des paramètres**.

4. ne pas confondre avec VAR qui permet la déclaration de variables
5. Si les variables avaient été déclarées au début (ligne 2), elles devenaient des **variables globales**

6.3.3 Le passage des paramètres

Il existe deux types de paramètres : les paramètres **formels**, présents lors de la déclaration (définition) de la procédure ou de la fonction et qui ne prennent aucune valeur et les paramètres **réels** qui, au moment de l'appel de la procédure ou de la fonction sont substitués aux paramètres formels. Cette substitution, appelée **passage des paramètres**, peut prendre deux formes : le **passage par valeur** et le **passage par adresse**.

Le passage par valeur

Dans un passage par valeur, le paramètre formel est considéré comme une variable locale dans le corps du sous-programme. Sa valeur est initialisée au début de chaque exécution du sous-programme avec la valeur du paramètre réel correspondant.

Il y a **recopie** de la valeur du paramètre réel dans le paramètre formel. Toutes les opérations qui sont effectuées sur le paramètre formel n'affectent que cette valeur locale.

```
procedure sp(... x: real ....);
```

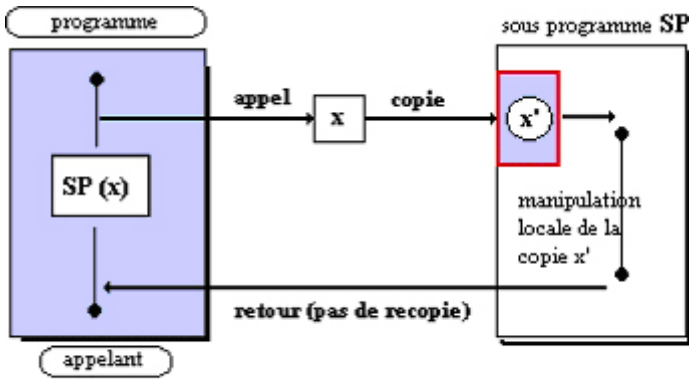


FIGURE 6.4 – Le passage par valeur

Le passage par adresse

Dans un passage par adresse, le paramètre formel est traité comme une variable dont l'**adresse** qui est transmise au moment de chaque appel est celle du paramètre réel correspondant.
Le paramètre réel et le paramètre formel sont la **même variable**. Le passage par adresse autorise toutes les modifications immédiatement sur le paramètre réel quelle que soit sa localisation.

```
procedure sp (.... var x : real);
```

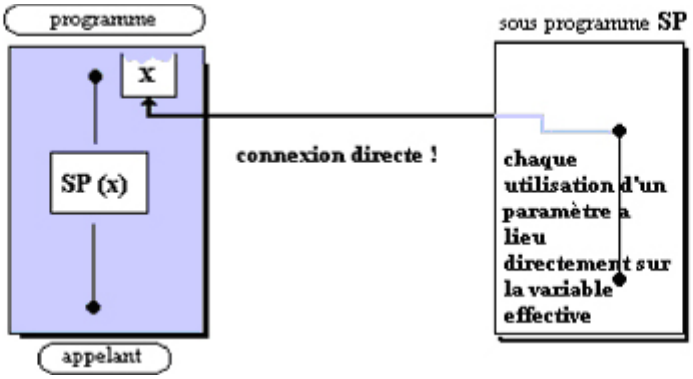


FIGURE 6.5 – Le passage par adresse

Le passage par adresse se reconnaît grâce au mot **VAR**.

Avantages et des inconvénients des passages par valeur et adresse

- **Passage par valeur**

  Avantages : - puisque le paramètre formel est une copie du paramètre réel, il est impossible pour le sous-programme de modifier le paramètre réel. C'est donc une **sécurité** ;
  - le paramètre réel peut être une constante.

  Inconvénients : - "lenteur" due à la recopie des données et doublement de la place mémoire occupée (mais convient bien pour des variables simples !) ;
  - ne permet pas de ressortir un résultat.

- **Passage par référence**

  Avantages : - rapidité d'accès aux données, moindre occupation mémoire puisqu'il ne s'agit que d'une adresse ;
  - permet de « ressortir » une valeur.

  Inconvénients : - puisque le sous-programme travaille en réalité sur le paramètre réel, il y a danger de modifier le paramètre réel de manière erronée ;
  - le paramètre réel ne peut être une constante.

Exemple

```
program Exemple;

procedure B1 (x : integer; var y : integer) ;
begin
   y := 10 * x;
end ;

procedure B2 (x : integer; y :integer) ;
begin
   y := 10 * x;
end ;

// Programme principal ou appelant
var a, b: integer ;
begin
   a := 100 ; b := 0 ;
   B1(a,b) ;
   writeln ('a = ', a, ' b = ', b);
   // a vaut 100 et b vaut 1000

   a := 100 ; b := 0;
   B2(a,b) ;
   writeln ('a = ', a, ' b = ', b);
   // a vaut 100 et b vaut 0

   readln;
end.
```

6.3.4 Variables locales, globales – Effet de bord

On appelle **variable locale** à un sous-programme (procédure ou fonction), une variable qui n'a de sens que pour ce sous-programme (ou pour tous les sous-programmes de niveau hiérarchique inférieur [6]).
Inversement, une **variable globale** est une variable qui a un sens en dehors du sous-programme.

6. Ce cas ne sera pas étudié dans ce cours.

Exemples et remarques

```
1  program Exemple;
2  var a,b : real;
3  procedure P1;
4  var c,d : real;
5  begin
6      .
7      .
8      .
9
10 end;
11 procedure p2;
12 var e,f : real;
13 begin
14     .
15     .
16     .
17 end;
18 // Programme principal
19 begin
20     .
21     .
22     .
23 end.
```

- les variables a et b (ligne 2) sont globales et sont accessibles aux deux procédures et au programme principal ;
- les variables c et d (ligne 4) ne sont accessibles que par la procédure P1 ;
- les variables e et f (ligne 12) ne sont accessibles que par la procédure P2 ;
- si une variable déclarée dans le corps d'un sous-programme (variable locale) porte le **même nom** qu'une variable globale, c'est la variable locale qui sera accessible dans le sous-programme ;

```
program Exemple;
var a,b : real;
procedure P1;
var a,c : real;
begin
    .   // a représente la variable locale
    .
    .

end;
// Programme principal
begin
    .
    .
    .
end.
```

- la modification d'une variable globale par un sous-programme est appelée **effet de bord** (”side effect”).
  Bien qu'il soit parfois approprié de laisser agir les sous-programmes sur le variables

globales, cette pratique doit être utilisée avec **énormément de prudence** et le programmeur débutant devrait toujours l'éviter. Considérons le programme suivant :

```
program effetDeBord;
var g : integer; // g est une variable globale
function F(x:integer) : integer;
begin
    g := g+1; // La variable globale est modifiée ici : effet de bord
    Result := x + g;
end;
// Programme principal ou appelant
begin
    g := 0;

    writeln(F(0)); // Affiche 1
    writeln(F(0)); // Affiche 2
    writeln(F(0)); // Affiche 3
    writeln(F(0)); // Affiche 4

    readln;
end.
```

La fonction F est appelée avec la **même** valeur de paramètre (0) et elle donne quatre **résultats différents**.
De plus, il faut savoir que les modifications *accidentelles* de variables globales par un sous-programme sont des erreurs qui sont souvent **difficiles à détecter**.

6.3.5 Exercices

1. Considérons le programme suivant :

```
program principal;
var a : real;
procedure p(x:real);
begin
   x := 1.0;
end;
begin
   a := 0.0;
   p(a);
   write(a);
end.
```

   (a) qu'imprime par *write(a)* ?

   (b) qu'imprime par *write(a)* si l'entête de la procédure est

```
procedure p(var x:real);
```

2. Considérons les deux procédures

```
procedure p1(x,y:integer);
var t : integer;
begin
   t := x;
   x := y;
   y := t;
end;
```

```
procedure p2(var x,y:integer);
var t : integer;
begin
   t := x;
   x := y;
   y := t;
end;
```

   Si $a = 3$ et $b = 5$,

   (a) quel est le résultat de l'appel p1(a,b) ?

   (b) quel est le résultat de l'appel p2(a,b) ?

3. Écrire un sous-programme qui calcule $1 + 2 + 3 + \cdots + N$.
4. Écrire un sous-programme qui calcule $N! = 1 \times 2 \times 3 \cdots \times N$.
5. Écrire un sous-programme qui teste si un entier est premier.
6. Écrire un sous-programme qui calcule a^n (a est un double (réel) ; n est un entier quelconque positif, négatif ou nul) sans utiliser Math.power.
7. Écrire un sous-programme qui calcule le pgcd (plus grand commun diviseur) de deux nombres entiers positifs.
   $pgcd(16, 24) = 8$
   $pgcd(80, 90) = 10$
8. Écrire un sous-programme qui réduit une fraction à sa plus simple expression :

$$\frac{39}{42} = \frac{13}{14}$$

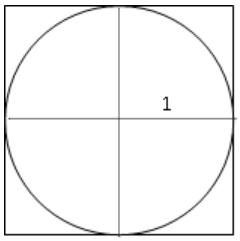
$$\frac{128}{12} = \frac{32}{3}$$

9. Écrire un sous-programme qui affiche tous les nombres de Pythagore compris entre 1 et N ($x^2 + y^2 = z^2$).
10. Écrire un sous-programme qui calcule le nombre de voyelles d'une chaîne de caractères.
11. Écrire un sous-programme qui calcule la somme des éléments d'un tableau d'entiers.
12. Écrire un sous-programme qui calcule le produit scalaire de deux tableaux (vecteurs)

$$T1 \bullet T2 = \sum_{i=1}^{n} T1[i]T2[i]$$

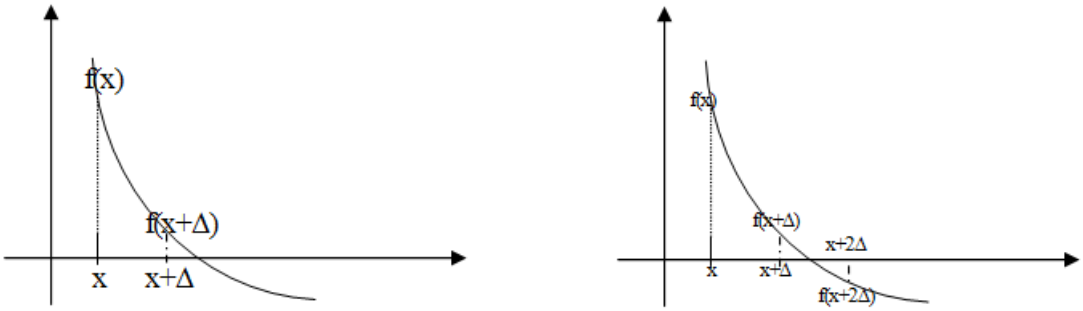
Par exemple, si T1 = 2,5,7,9 et T2 = 4,2,3,1 alors le produit scalaire vaut $2 \times 4 + 5 \times 2 + 7 \times 3 + 9 \times 1 = 48$

13. Écrire un sous-programme qui calcule le minimum des éléments d'un tableau de réels (double).
14. Écrire un sous-programme qui génère aléatoirement un tableau (préciser l'énoncé).
15. Écrire un sous-programme qui dit combien de fois un entier se trouve dans un tableau d'entiers.
16. Écrire un sous-programme qui trie un tableau.
17. Écrire la recherche dichotomique grâce à une fonction ; le tableau sera au préalable trié grâce à une autre fonction (ou procédure).
18. L'aire d'un cercle de rayon 1 est π et l'aire d'un carré qui contient exactement ce cercle est 4. Ainsi, si un grand nombre de points sont choisis aléatoirement dans la partie supérieure droite du carré, la proportion de ces points qui seront dans le cercle est approximativement $\pi/4$. Calculer une valeur approximative de π.



19. Programmer le jeu du « pendu ».
20. Programmer le jeu « mastermind ».
21. Écrire un programme qui génère deux nombres aléatoires entre 0 et 100 et choisit une opération au hasard (+,-,*,/). Le programme affiche l'opération et lit le résultat. Si la réponse donnée est incorrecte, il donne une deuxième chance. Si elle est toujours incorrecte, le programme donne la bonne réponse. Tous les 10 exercices, il rappelle le nombre de bonnes réponses.

22. Trouver une racine d'une équation grâce à la méthode des intervalles. Commençant en un point arbitraire, f(x) est évalué en une série de points qui sont distants d'une valeur Δ donnée. Si on trouve que le signe de f(x) a changé entre deux évaluations successives, c'est qu'on a passé une racine.



(a) (b)

(a) Entre $x \; et \; x + \Delta$, la fonction f(x) ne change pas de signe

$$f(x) > 0 \text{ et } f(x + \Delta) > 0$$

c'est donc qu'il n'y a aucune racine entre x et $x + \Delta$.

(b) Entre $x + \Delta$ et $x + 2\Delta$, la fonction f(x) change de signe

$$f(x + \Delta) > 0 \text{ et } f(x + 2\Delta) < 0$$

c'est donc qu'il y a une racine entre $x + \Delta$ et $x + 2\Delta$. Dans ce cas, on recommence avec Δ diminué de moitié et dans l'autre sens (en réalité $\Delta = -\Delta/2$)

CHAPITRE 7

LES FICHIERS

Il est souvent nécessaire de stocker des informations de façon **permanente**. Le seul moyen envisageable, mis à part l'utilisation d'une base de données, est l'utilisation de fichiers.

Les fichiers sont universellement utilisés pour stocker des informations, et leur utilisation est permise de plusieurs manières en Pascal.

7.1 Différents types de fichiers

7.1.1 Fichiers texte

Les fichiers texte comportent du texte brut. Le texte contenu dans ce genre de fichiers est réparti en lignes. Pour information, chaque ligne se termine, comme dans tous les fichiers texte, par un marqueur de fin de ligne qui est constitué d'un ou de deux caractères spéciaux : le retour chariot et éventuellement le saut de ligne (sous les environnements Windows, les deux caractères sont généralement présents, contrairement aux environnements de type UNIX qui favorisent le retour chariot seul).

Ces fichiers sont adaptés à des tâches très utiles :

- stocker du texte simple entré par l'utilisateur (fichiers TXT)
- écrire des fichiers de configuration (entre autre des fichiers INI)
- générer des fichiers lisibles par d'autres applications, comme des fichiers HTML (pages web), CSV (texte réparti en colonnes).
- écrire des programmes dans différents langages (PAS, PHP, JAVA, C, ...)

7.1.2 Fichiers à accès direct

Ces fichiers sont structurés en éléments ou enregistrements (record) dont le type peut être quelconque. Les éléments d'un fichier à accès direct se suivent logiquement, mais pas forcément physiquement ; ils ont tous la même taille. Le terme accès direct signifie qu'il est possible d'accéder directement à n'importe quel élément, pour autant que l'on spécifie son rang.

7.1.3 Fichiers binaires ou non typés

Ce dernier genre de fichier est en fait un genre universel, car tous les types de fichiers, sans exception, peuvent être créés avec ce genre de fichier. Mais cela a un prix : les facilités offertes en lecture/écriture par les fichiers texte ou les fichiers séquentiels n'existent pas pour ce type de fichier.

7.2 Manipulation des fichiers texte

7.2.1 Ouverture et fermeture d'un fichier texte

Un fichier texte se manipule par l'intermédiaire d'une variable de type *Text*. La manipulation d'un fichier passe par trois étapes obligatoires : deux avant l'utilisation, et une après. Voici la déclaration de la variable fichier :

```
var FichTest: Text;
```

1. La **première étape**, avant utilisation, est l'étape d'**assignation**. Elle consiste à associer à la variable-fichier le nom du fichier que nous voulons manipuler. Cette assignation se fait par l'appel d'une procédure très simple nommée *Assign* qui admet deux paramètres. Le premier est la variable-fichier, le second est le nom du fichier à manipuler. Voici l'instruction qui assigne le fichier *c :\test.txt* à la variable-fichier *FichTest* déclarée ci-dessus :

   ```
   Assign(FichTest, 'c:\test.txt');
   ```

2. La **deuxième étape** consiste à ouvrir le fichier. Cette ouverture peut se faire suivant trois modes différents suivant ce qu'on a besoin de faire pendant que le fichier est ouvert. La méthode qu'on emploie pour ouvrir le fichier détermine ce mode d'ouverture. Les trois modes possibles sont la lecture seule (écriture impossible), l'écriture seule (lecture impossible), soit simplement l'ajout du texte à la fin du fichier (ajout seul). **Il n'est pas possible de pouvoir lire et écrire à la fois dans un fichier texte**.

   - Si on souhaite lire, il faut utiliser la procédure *Reset*. Si le fichier n'existe pas, une erreur est générée (”File not found”). L'écriture n'est pas possible avec ce mode d'ouverture.
   - Si on souhaite écrire, il faut utiliser la procédure *Rewrite*. Si le fichier existe, il sera écrasé. La lecture n'est pas possible avec ce mode d'ouverture.
   - Si on souhaite juste ajouter du texte dans un fichier texte, il faut l'ouvrir en utilisant la procédure *Append*. Cette procédure ouvre le fichier et permet d'y écrire seulement. Ce qui est écrit sera ajouté à la fin du fichier.

   Ces trois procédures s'utilisent de la même manière : on les appelle en donnant comme unique paramètre la variable-fichier à ouvrir. Voici l'instruction qui commande l'ouverture du fichier *c :\test.txt* en lecture seule :

   ```
   Reset(FichTest);
   ```

   Pour ouvrir le même fichier en écriture seule, il suffirait de substituer “Rewrite” (ou “Append”) à “Reset”.

3. La **troisième étape** est la fermeture du fichier. La procédure *Close* (qui accepte un seul paramètre de type variable-fichier) ferme un fichier ouvert indifféremment avec "Reset", "Rewrite" ou "Append". Voici l'instruction qui ferme le fichier :

```
Close(FichTest);
```

Voici donc une procédure qui ouvre un fichier en lecture seule et le ferme aussitôt.

```
procedure OuvrirFichier;
var FichTest: Text;
begin
   Assign(FichTest, 'C:\Test\Fich.txt');
   Reset(FichTest);
   { lecture possible dans le fichier ici }
   Close(FichTest);
end;
```

Existence d'un fichier Il est possible d'améliorer cette procédure d'ouverture, notamment en contrôlant l'existence du fichier avant son ouverture, car un fichier inexistant provoquerait une erreur gênante qu'il est possible d'éviter facilement. Voici une méthode pour tester l'existence d'un fichier :

la fonction *FileExists* définie dans l'unité *SysUtils* teste si un fichier existe, le résultat de cette fonction est donc un booléen.

```
program existenceFichier;
{ Ce programme illustre l'utilisation de FileExists }
uses sysutils;
var fich : text;
   nomFichier : String;
begin
   write('Entrez le chemin et le nom d''un fichier : ');
   readln(nomFichier);
   if FileExists(nomFichier) then    // = true inutile
   begin
      reset(nomFichier); // On est certain que le fichier existe
      ⋮
   end;
end.
```

```
Entrez le chemin et le nom d'un fichier : c:\56TTi\UProcedure.pas
Le fichier existe.
```

7.2.2 Lecture depuis un fichier texte

La lecture se fait au choix par la procédure *Read* ou *Readln* ("READ LiNe", "lire ligne" en anglais). La procédure Readln est la plus simple à utiliser car elle permet de lire une ligne entière d'un fichier (pour rappel, un fichier texte est découpé en lignes). La procédure Read, quant à elle, permet de lire de différentes manières : caractère par caractère, ou petit morceau par petit morceau. Cette dernière technique est à éviter car elle prend plus de temps que la lecture ligne par ligne, et tout ce que permet la procédure Read peut être réalisé sur le texte lu au lieu d'être réalisé sur le texte à lire. La lecture ligne par ligne (*Readln*) sera donc la seule étudiée ici.

Lorsqu'on ouvre un fichier texte, la "position" dans ce fichier est fixée à son début. La "position" d'un fichier détermine à quel endroit aura lieu la prochaine opération sur ce fichier. Pour lire une ligne d'un fichier, on utilise donc Readln. Readln est une procédure très spéciale dans le sens où son nombre de paramètres est a priori indéterminé. Le premier paramètre est la variable-fichier dans laquelle on désire lire. Les paramètres suivants sont des variables permettant de stocker ce qui a été lu. Dans la pratique, **on se limite très souvent à une seule variable de type chaîne**. Voici l'instruction qui lit une ligne du fichier associé à la variable-fichier TestFile et qui la stocke dans la variable tmpS de type string :

```
Readln(FichTest, tmpS);
```

Il va de soi que pour que cette instruction fonctionne, il faut que le fichier soit ouvert. Readln lit une ligne depuis la "position" en cours dans le fichier, jusqu'aux caractères de fin de ligne (ceux-ci sont éliminés), puis fixe la nouvelle "position" au début de la ligne suivante (sous réserve qu'elle existe, nous allons parler de l'autre cas un peu plus bas). L'appel suivant à Readln lira la ligne suivante du fichier, et ainsi de suite jusqu'à ce qu'il n'y ait plus de ligne après la position dans le fichier, c'est-à-dire lorsqu'on a atteint la fin du fichier. Pour tester cette condition, on doit faire appel à une fonction nommée *Eof* ("End Of File", "Fin de fichier" en anglais). Cette fonction accepte en unique paramètre une variable-fichier et renvoie un booléen qui indique si la position de fichier est la fin de ce dernier, c'est-à-dire que lorsque Eof renvoie True, il n'y a plus de ligne à lire dans le fichier.

La plupart du temps, lorsqu'on lit l'ensemble des lignes d'un fichier texte, il faut utiliser une boucle while. Cette boucle permet de lire les lignes une par une et de tester à chaque fois si la fin du fichier est atteinte. Voici un exemple d'une telle boucle while :

```
while not Eof(FichTest) do
   Readln(FichTest, tmpS);
```

La boucle ci-dessus lit un fichier ligne par ligne jusqu'à la fin du fichier. Lorsque celle-ci est atteinte, Eof devient vrai et la condition de continuation de la boucle n'est plus respectée, et la lecture s'arrête donc d'elle-même.

Voici notre procédure OuvrirFichier encore améliorée. Elle lit maintenant l'intégralité du fichier transmis et ne fait rien avec ce qui a été lu :

```
procedure OuvrirFichier(chemin : string);
uses sysutils;
var
   FichTest: Text;
   tmpS: string;
begin
   if not FileExists(chemin) then
      Exit; // Arrêt si non existence
   Assign(FichTest, chemin);
   Reset(FichTest);
   while not Eof(FichTest) do
      Readln(FichTest, tmpS);
   Close(FichTest);
end;
```

7.2.3 Écriture dans un fichier texte

L'écriture dans un fichier texte s'effectue également ligne par ligne grâce à la procédure *Writeln*, ou morceau par morceau avec la procédure *Write*. La procédure Writeln s'utilise comme Readln, mis à part qu'elle effectue une écriture à la place d'une lecture. L'écriture se fait à la position actuelle, qui est définie au début du fichier lors de son ouverture par "Rewrite", et à la fin par "Append". Writeln ajoute la chaîne transmise sur la ligne actuelle et insère les caractères de changement de ligne, passant la position dans le fichier à la ligne suivante. Ainsi, pour écrire tout un fichier, il suffira d'écrire ligne par ligne en appelant autant de fois Writeln qu'il sera nécessaire. Voici une procédure qui ouvre un fichier *c :\test.txt* en tant que texte, et qui écrit deux lignes dans ce fichier. Après l'exécution de la procédure, vous pourrez contrôler le résultat en ouvrant ce fichier dans le bloc-notes par exemple. Attention au nom du fichier car **l'ouverture avec Rewrite ne teste pas l'existence du fichier** et il est donc possible d'écraser par mégarde un fichier important.

```
procedure TestEcriture(Fich: string);
var F: Text;
begin
   Assign(F, Fich);
   Rewrite(F);
   Writeln(F, 'ceci est notre premier fichier texte');
   Writeln(F, 'rien de difficile!');
   Close(F);
end;
```

Dans l'exemple ci-dessus, le fichier texte est ouvert en écriture seule par Rewrite, puis deux écritures successives d'une ligne à chaque fois sont effectuées. Le fichier est ensuite normalement refermé. L'écriture est beaucoup plus simple que la lecture puisqu'on n'a pas à se soucier de la fin du fichier.

7.2.4 Exemple

```
procedure Ajout;
(* Permet d'ajouter du texte (Bonjour) dans un fichier Text *)
uses sysutils;
var f: Text;
    ch: string;
    chemin: string;
begin
    chemin := 'c:\temp\test.txt';

    Assign(f,chemin);
    if not FileExists(chemin) then
       Rewrite(f)      // Création du fichier
    else
       Append (f);     // Ouverture du fichier pour y ajouter du texte

    ch := 'Bonjour';
    writeln(f, ch);   // Ajoute ch et un fin de ligne
    Close(f);
end;

procedure Lire;
(* Permet de lire le contenu d'un fichier Text *)
uses sysutils;
var f: Text;
    ch: string;
    chemin: string;
begin
    chemin := 'c:\Fichiers\Fich.dat';
    if not FileExists(chemin) then
       Exit; // Arrêt si non existence
    Assign(f,chemin);
    Reset (f); // Permet d'ouvrir le fichier pour y lire
    // Tant que la fin de fichier n'est pas atteinte
    while Eof(f) = false do // ou while not Eof(f)
    begin
       readln(f, ch);      // Lit une chaîne du fichier
       writeln(ch);        // et l'écrit à l'écran
    end;
    Close(f);
end;

procedure Vider;
(* Vide un fichier --> recrée le fichier *)
var f: Text;
begin
    Assign(f,'c:\temp\test.txt');
    Rewrite(f);      // Création du fichier
    Close(f);
end;
```

7.2.5 Effacement d'un fichier

L'effacement d'un fichier texte s'effectue grâce à la procédure *Erase*. Le fichier doit être assigné mais pas ouvert.

```
Assign(f,chemin);
Erase (f);
```

7.2.6 Renommage d'un fichier

Le renommage d'un fichier texte s'effectue grâce à la procédure *Rename*. Le fichier doit être assigné mais pas ouvert.

```
Assign(f,chemin);
Rename(f, nouveau_nom);
```

7.2.7 Exercices

1. Compter le nombre de lignes d'un fichier texte.
2. Mettre tous les caractères d'un fichier texte en majuscules.
3. Lire un fichier texte et y remplacer les voyelles par #.
4. Dans un fichier HTML, remplacer les balises gras par italique.
5. Rechercher le nombre de fois qu'apparaît un mot donné dans un fichier texte.
6. Écrire un programme qui affiche le contenu d'un fichier texte en faisant précéder chaque ligne par son numéro.
7. Créer un fichier texte appelé exercice.txt et y stocker des valeurs (de type string) récupérées depuis le clavier jusqu'à la lecture de la chaîne 'FIN' (peut importe la casse, 'fin' ou 'fIn'). La chaîne de caractère, 'FIN' en l'occurrence ne sera pas stockée dans le fichier.
8. Compter le nombre de mots d'un fichier texte.
9. Calculer la fréquence d'apparition des lettres dans un fichier texte.
10. Écrire un programme qui affiche le contenu d'un fichier texte en ignorant les lignes de commentaires et en supprimant les blancs en début de ligne.
    - Les caractères "blancs" sont les espaces et les tabulations [1].
    - Les lignes dont le premier caractère non blanc est un § seront considérées comme des commentaires.

    Ainsi, le fichier texte suivant :

```
§ Une famille
  Raymonde
   Robert
   § et leurs enfants:
       Jules
Jim
§ fin
```

    sera affiché comme ceci :

```
Raymonde
Robert
Jules
Jim
```

1. Chr(9) est le caractère qui représente une tabulation

11. Ajouter des élèves dans un fichier texte. On suppose que les lignes du fichier ressemblent à celles-ci :

    ```
    reference:nom:prenom
    ```

    Par exemple le fichier pourrait contenir :

    ```
    12:Bob:Morane
    3:Bruce:Wayne
    18:Corben:Dallas
    ```

12. Ajouter des élèves dans un fichier texte. On suppose que les données sont de largeur fixe : 3 caractères pour la référence, 20 pour le prénom et 15 pour le nom.

    Par exemple le fichier pourrait contenir (sans la première ligne qui est un "repère") :

    ```
    123123456789012345678901234567890123 45
    12  Bob                 Morane
    3   Bruce               Wayne
    18  Corben              Dallas
    ```

7.3 Manipulation des fichiers à accès direct

Voici un exemple de déclarations liées au traitement d'un fichier à accès direct :

```
type element =
   record
      nom : string[12];
      age : integer;
   end;
var personne : element;
   f : file of element;
```

Le schéma suivant illustre l'organisation logique des premiers enregistrements du fichier f :

Rang	0	1	2	3
Nom	Durant	Mercier	César	Alex
Age	24	44	37	67

Chaque enregistrement d'un fichier à accès direct peut être accédé en spécifiant son rang. Un pointeur de fichier mémorise le rang de l'enregistrement concerné par la prochaine opération de lecture ou d'écriture. Lorsqu'un fichier est ouvert, son pointeur indique l'enregistrement de rang zéro, c'est-à-dire le premier enregistrement. Après chaque opération de lecture ou d'écriture, le pointeur de fichier est incrémenté du nombre d'enregistrements lus ou écrits. Un enregistrement est l'unité minimale de transfert d'information entre un fichier et un programme. Le contenu des fichiers à accès direct est stocké sous forme binaire compactée et n'est donc pas directement affichable à l'écran. Il est également important de savoir que l'on peut accéder de manière séquentielle à un fichier à accès direct. Très souvent les éléments d'un fichier à accès direct sont de type enregistrement.

7.3.1 Déclaration du type de fichier à accès direct

La déclaration d'un fichier à accès direct est effectuée à l'aide des mots réservés *file of* :

```
type client = record
   code : string[10];
   nom : string[40];
   solde : real;
   premiere_facture : integer;
   derniere_facture : integer;
end;
var fichier_client : file of client;
   un_client : client;
   ch : char;
```

En plus d'une variable de type fichier, il faut définir une variable dont le type est celui des éléments de ce fichier. Cette dernière *un_client* est utilisée comme paramètre par les procédures de lecture et d'écriture. Elle tient lieu de variable intermédiaire, ou de mémoire tampon.

7.3.2 Assignation de fichier

La procédure *Assign* remplit le même rôle que pour les fichiers texte :

```
assign (fichier_client, 'CLIENT.DAT');
```

7.3.3 Ouverture de fichier

Cette opération est effectuée par l'une des procédures *reset* ou *rewrite*, comme pour les fichiers de type texte :

```
rewrite (fichier_client); { le fichier peut exister }
reset (fichier_stock); { le fichier doit exister }
```

L'emploi de la procédure *reset* nécessite l'existence préalable du fichier. En revanche l'utilisation de la procédure *rewrite* crée et ouvre inconditionnellement un fichier. Si un fichier portant le même nom existait déjà, son contenu serait perdu. Il convient donc d'être prudent lors de la création d'un fichier à l'aide de cette procédure.

Contrairement aux fichiers texte, l'ouverture avec **Reset permet la lecture et l'écriture dans un fichier**.

7.3.4 Positionnement à l'intérieur d'un fichier

A chaque fichier utilisé dans un programme est associé un pointeur permettant au système et à l'utilisateur de connaître l'emplacement du prochain élément accessible. La procédure

```
seek (nom_de_fichier, no)
```

permet de positionner le pointeur associé au fichier nom_de_fichier sur l'enregistrement numéro *no*, en vue d'une opération de lecture ou d'écriture. Il faut se rappeler que le premier enregistrement d'un fichier porte le numéro zéro.

7.3.5 Instructions de lecture et d'écriture

La procédure de lecture est

```
read (nom_de_fichier, liste_elements)
```

et celle d'écriture est

```
write (nom_de_fichier, liste_elements)
```

où liste_elements est une liste de variables du même type que les éléments du fichier. Il est important de savoir qu'à chaque écriture et à chaque lecture le pointeur associé au fichier est automatiquement incrémenté de manière à être positionné sur l'enregistrement suivant. Partant de ce principe, la modification du contenu de l'enregistrement numéro 5 d'un fichier implique, par exemple, les opérations suivantes :

```
seek (fichier_client, 5);
read (fichier_client, un_client);
   ...{ modification des données contenues dans un_client }
seek (fichier_client, 5);
write (fichier_client, un_client);
```

7.3.6 Enregistrement courant et taille d'un fichier

La fonction

```
FilePos (nom_de_fichier)
```

fournit le rang de l'enregistrement courant. La fonction

```
FileSize (nom_de_fichier)
```

donne la taille du fichier, exprimée en nombre d'éléments. Le fragment de programme suivant permet de déterminer si un fichier est vide :

```
if filesize (f) = 0 then
   writeln ('Fichier vide');
```

7.3.7 Fin de fichier

La fonction booléenne

```
Eof (nom_de_fichier)
```

indique si la fin du fichier nom_de_fichier est atteinte, comme pour les fichiers texte

7.3.8 Fermeture de fichier

La fermeture d'un fichier à accès direct s'effectue de la même manière que pour un fichier de type texte :

```
close (fichier_client);
```

7.3.9 Effacement et renommage d'un fichier

L'effacement et le renommage s'utilisent de la même manière que pour les fichiers texte.

7.3.10 Troncature d'un fichier

Cette procédure permet de couper à l'*endroit actuel* le contenu du fichier.

```
Truncate (fichier_client);
```

7.3.11 Exemples

1. Lire le contenu d'un fichier d'enregistrements et afficher les renseignements à l'écran.

```
program LireFileRecord;
uses sysutils;
type TEleves =
        record
                numero : Integer;
                nom : String[20];
                prenom : String[20];
                sexe : char;
                natio : String[3];
                classe : String[4];
                localite : String[30];
                points : integer;
        end;
var  unEleve : TEleves;
     fichEleves : file of TEleves;
     chemin : string;
begin
     chemin := 'C:\Users\Henrotte\Dropbox\Travaux\dev_Pascal\Record\ElevesNumLim
     if not FileExists(chemin) then
     begin
          writeln('Le chemin du fichier n''est pas correct ou alors le fichier n
          writeln('Fermeture du programme.');
          readln;
          exit;
     end;

     assign(fichEleves,chemin);
     reset(fichEleves);
     while eof(fichEleves) = false do
     begin
          read(fichEleves,unEleve);
          writeln(unEleve.numero:3,' ',
                  unEleve.nom,StringOfChar(' ',20-length(unEleve.nom)),
                  unEleve.prenom,StringOfChar(' ',20-length(unEleve.prenom)),
                  unEleve.sexe,StringOfChar(' ',2-length(unEleve.sexe)),
                  unEleve.natio,StringOfChar(' ',4-length(unEleve.natio)),
                  unEleve.classe,StringOfChar(' ',5-length(unEleve.classe)),
                  unEleve.localite,' ',StringOfChar(' ',30-length(unEleve.localite)),
                  unEleve.points);

     end;
     close(fichEleves);
     readln;
end.
```

2. L'exemple qui suit
   - copie un fichier d'élèves dans un tableau (le fichier est supposé existé et remplit d'élèves) ;
   - trie le tableau sur le champ *nom* ;
   - affiche le tableau qui vient d'être trié ;
   - recopie le tableau dans un autre fichier.

```
program ProgEl;
uses sysutils; // pour fileExists

type TEleve = record
          nom : String[20];
          prenom : String[20];
          sexe : char;
          nationalite : String[3];
          classe : String[4];
          localite : String[30];
          points : Integer;
       end;
var eleve : TEleve;   // un élève
    memoire : TEleve; // pour le tri et contient un élève
    f : file of TEleve;   // fichier de départ
    f_tri : file of TEleve; // fichier trié
    tabEleves : array[1..1000] of TEleve;   // pas plus de 1000 élèves
    nbEleves : integer;   // le nombre d'élèves
    i : integer;
    echange : boolean;    // pour le tri

begin
    (* Arrêt si le fichier n'existe pas *)
    if not fileExists('c:\56TTi\CreeFich\Eleves.dat') then
    begin
        write ('Fichier inexistant!');
        readln;
        halt;
    end;
```

```
    (* Copie du fichier dans le tableau *)
    assign(f,'c:\56TTi\CreeFich\Eleves.dat');
    reset(f);
    nbEleves := 0;
    while not eof(f) do
    begin
       read(f,eleve); // lire un enregistrement
       nbEleves += 1; // incrémenter l'indice
       tabEleves[nbEleves] := eleve; // copier l'enregistrement dans le tableau
    end;
    close(f);

    (* Tri ''bulle'' *)
    repeat
       echange := false;
       for i := 1 to nbEleves-1 do
       begin
          if tabEleves[i].nom > tabEleves[i+1].nom then
          begin
             memoire := tabEleves[i];
             tabEleves[i] := tabEleves[i+1];
             tabEleves[i+1] := memoire;
             echange := true;
          end;
       end;
    until echange = false;

    (* Affichage du tableau *)
    for i:=1 to nbEleves do
    begin
       with tabEleves[i] do
       begin
          writeln(nom, ' ',prenom,' ',sexe,' ',nationalite,' ',
                classe,' ',localite,' ',points);
       end;
    end;

    (* Copie du tableau dans un fichier qu'on crée *)
    assign(f_tri,'c:\56TTi\CreeFich\ElevesTRI.dat');
    rewrite(f_tri);
    for i:=1 to nbEleves do
       write(f_tri,tabEleves[i]);
    close(f_tri);

    readln;
end.
```

3. Le but de l'exemple ci-dessous est de remplir un fichier d'élèves mais **sans devoir encoder les renseignements soi-même**. En général, il est facile de transférer les données d'un logiciel ou d'une base de données vers un format texte.
   On suppose qu'on dispose d'un fichier texte dont la structure est

   Nom%Prénom%Sexe%Nationalité%Classe%Localité%Points%

   Chaque renseignement est séparé du suivant par le caractère %
   Par exemple,

```
COLPIN%Cyril%M%F%5Pb%LONGUYON%26%
FROGNET%Victorien%M%B%5Pb%IRE-LE-SEC%94%
GEORGES%Bastien%M%B%5Pb%ROBELMONT%76%
GOMEZ%Michel%M%F%5Pb%CHATILLON%21%
LUQUER%Christopher%M%F%5Pb%LUXEMBOURG%61%
WESOLOWSKI%Arnaud%M%F%5Pb%MONDELANGE%50%
ALEXANDRE%Valentin%M%B%5Pc%MEIX-DEVANT-VIRTON%96%
CHERIAK%Mustapha%M%DZ%5Pc%SAINT NICOLAS-DE-PORT%57%
COLLIGNON%Thomas%M%B%5Pc%SAINT-VINCENT%64%
FORGET%Dorian%M%B%5Pc%PALISEUL%23%
⋮
```

   On voudrait créer un fichier d'enregistrements qui reprend le contenu du fichier texte ci-dessus.

```
program Transfert;
uses sysutils;
type TEleves =     // Renseignements pour chaque élève
record
nom : String[20];
prenom : String[20];
sexe : char;
natio : String[3];
classe : String[4];
localite : String[30];
point : integer;
end;

var f : text;
unEleve : TEleves;
fichEleves : file of TEleves;
chemin,ch : string;
strPoint : String;    // Points transformés en chaîne
position : integer;   // Trouve le prochain caractère %
code : Integer;
```

```
begin
chemin := 'C:\dev_Pascal\Text_Record\Eleves.txt';
// Teste l'existence du fichier, arrêt s'il n'existe pas
if not FileExists(chemin) then
begin
writeln('Le chemin du fichier n''est pas correct ou alors le fichier n''existe
writeln('Fermeture du programme.');
readln;
exit; // Quitte
end;

// Ouverture
assign(f,chemin);
reset(f);
assign (fichEleves,'C:\dev_Pascal\Text_Record\Eleves.rec');
rewrite(fichEleves);

while eof(f) = false do
begin
readln(f,ch);

// Extraction du nom
position := pos('%',ch);     // Recherche de la position du %
unEleve.nom := copy(ch,1,position-1);//Extraction du nom jusqu'au % non compris
Delete(ch,1,position);   // Effacement du nom et du %

// Extraction du prénom
position := pos('%',ch);
unEleve.prenom := copy(ch,1,position-1);
Delete(ch,1,position);

// Extraction du sexe
unEleve.Sexe :=ch[1];
Delete(ch,1,2);

// Extraction de la nationalité
position := pos('%',ch);
unEleve.natio := copy(ch,1,position-1);
Delete(ch,1,position);

// Extraction de la classe
position := pos('%',ch);
unEleve.classe := copy(ch,1,position-1);
Delete(ch,1,position);

// Extraction de la localité
position := pos('%',ch);
unEleve.localite := copy(ch,1,position-1);
Delete(ch,1,position);

// Extraction des points (attention, c'est une chaîne dans le fichier texte)
position := pos('%',ch);
strPoint := copy(ch,1,position-1);
Val(strPoint,unEleve.point,code);     // Transformation en numérique
Delete(ch,1,position);
```

```
// Ecriture de l'enregistrement dans le fichier d'enregistrements
write(fichEleves,unEleve);
end;

close(f);
close(fichEleves);

readln;
end.
```

Remarque *with* : au lieu d'écrire

```
eleve.nom := ···;
eleve.prenom := ···;
   ⋮
```

on peut écrire

```
with eleve do
begin
   nom := ···;
   prenom := ···;
      ⋮
end;
```

7.3.12 Exercice : gestion simplifiée d'un fichier d'élèves

Supposons qu'un fichier contienne des enregistrements dont la structure est :

```
type TEleves =
   record
      numero : Integer;
      nom : String[20];
      prenom : String[20];
      sexe : char;
      natio : String[3];
      classe : String[4];
      localite : String[30];
      points : integer;
   end;
```

1. Afficher tous les renseignements de tous les élèves.
2. Rechercher un élève connaissant son nom (il peut y en avoir plusieurs).
3. Rechercher un élève connaissant son numéro.
4. Calculer le nombre de filles.
5. Trouver l'élève dont le champ "points" est maximum.
6. Supprimer un élève connaissant son numéro.
7. Insérer un nouvel élève (son numéro sera 1 de plus que le dernier numéro).
8. Modifier les renseignements d'un élève.
9. Trier le fichier par ordre alphabétique (le plus simple est de passer par un tableau).

10. Trier le fichier par classe.
11. En supposant que le fichier soit trié par classe,
    (a) compter le nombre d'élèves par classe (**algorithme de rupture**);
    (b) calculer les points moyens par classe.
12. ⋮

Remarques

- ne pas utiliser de tableaux (sauf pour les tris)
- on peut utiliser Seek, FilePos et FileSize

7.4 Fichiers non typés

Les fichiers non typés se déclarent à l'aide du mot réservé *file* :

```
var fichier: file; { et non: file of... }
```

Une variable de type file permet d'accéder à un fichier quelconque, sans connaître sa structure, ni son contenu. Les opérations de lecture et d'écriture sont effectuées par les procédures *BlockRead* et *BlockWrite*. Les opérations d'assignation et de fermeture sont les mêmes que pour les fichiers à accès direct ou de type texte.

```
Program FichierSansType;
(*
   Ce programme illustre BlockRead et BlockWrite
   en copiant un fichier dans un autre.
*)
Var F_In , F_Out : File;    // Fichiers entrée et sortie
   NumRead, NumWritten : Word;
   Buf : Array [1..2048] of byte;
   Total : Longint;
begin
   Assign (F_In, 'C:\56TTi\project.exe');
   Assign (F_Out, 'C:\56TTi\copie.exe');
   Reset (F_In, 1);  // 1 = taille d'un enregistrement
   Rewrite (F_Out, 1);

   Total := 0;
   Repeat
      BlockRead (F_In, Buf, Sizeof(Buf), NumRead);
      BlockWrite (F_Out, Buf, NumRead, NumWritten );
      inc (Total, NumWritten);
   Until (NumRead = 0) or (NumWritten <> NumRead);
   Write (Total, ' caractères ont été copiés!');

   Close (F_In);
   Close (F_Out);

   Readln;
end.
```

CHAPITRE 8

COMPLÉMENTS

8.1 L'unité CRT

8.1.1 Quelques constantes

Black = 0	Green = 2	LightRed = 12
Blink = 128	LightBlue = 9	Magenta = 5
Blue = 1	LightCyan = 11	Red = 4
Brown = 6	LightGray = 7	White = 15
Cyan = 3	LightGreen = 10	Yellow = 14
DarkGray = 8	LightMagenta = 13	

8.1.2 Quelques fonctions ou procédures

ClrEol	KeyPressed	TextColor
ClrScr	ReadKey	WhereX
Delay	Sound	WhereY
GotoXY	TextBackground	Window

ClrEol

ClrEol clears the current line, starting from the cursor position, to the end of the window. The cursor doesn't move.

ClrScr

ClrScr clears the current window (using the current colors), and sets the cursor in the top left corner of the current window.

Delay

Delay waits a specified number of milliseconds. The number of specified milliseconds is an approximation, and may be off a lot, if system load is high.

GotoXY

GotoXY positions the cursor at (X,Y), X in horizontal, Y in vertical direction relative to the origin of the current window. The origin is located at (1,1), the upper-left corner of the window.

```
Program Example;
uses Crt;
begin
   ClrScr;
   GotoXY(10,10) ;
   Write ('10,10') ;
   GotoXY(70,20) ;
   Write ('70,20') ;
   GotoXY(1,22) ;
end.
```

KeyPressed

Keypressed scans the keyboard buffer and sees if a key has been pressed. If this is the case, True is returned. If not, False is returned.
The Shift, Alt, Ctrl keys are not reported. The key is not removed from the buffer, and can hence still be read after the KeyPressed function has been called.

ReadKey

ReadKey reads 1 key from the keyboard buffer, and returns this. If an extended or function key has been pressed, then the zero ASCII code is returned. You can then read the scan code of the key with a second ReadKey call.

```
Program Example;
uses Crt;
var ch : char;
begin
   writeln('Press Left/Right/Up/Down,Esc=Quit');
   repeat
      ch := ReadKey;
      case ch of
         #0 :  begin
                  ch := ReadKey; {Read ScanCode }
                  case ch of
                     #75 : writeln('Left');
                     #77 : writeln('Right');
                     #80 : writeln('Down');
                     #72 : writeln('Up');
                  end;
               end;
         #27 : writeln('ESC');
      end;
   until ch=#27 {Esc}
end.
```

On trouve la liste de tous les "scancodes" aux pages 720 et 721 du fichier RTL.PDF (cf. page 134).

Sound

Sound(hz) sounds the speaker at a frequency of hz. Under Windows, a system sound is played and the frequency parameter is ignored.

TextBackground

TextBackground(color) sets the background color to *color*. *color* can be one of the predefined color constants.

```
Program Example;
uses Crt;
begin
   TextColor (White) ;
   Writeln ('Ecriture en blanc sur le fond par défaut');
   TextBackground (Green) ;
   Writeln ('Ecriture en blanc sur un fond vert');
   TextBackground (Brown);
   Writeln ('Ecriture en blanc sur un fond brun') ;
   TextBackground (Black) ;
   Writeln ('Retour à un fond noir' ) ;
end.
```

TextColor

TextColor(color) sets the foreground color to *color*. *color* can be one of the predefined color constants.

WhereX

WhereX returns the current X-coordinate of the cursor, relative to the current window. The origin is (1,1), in the upper-left corner of the window.

WhereY

WhereY returns the current Y-coordinate of the cursor, relative to the current window. The origin is (1,1), in the upper-left corner of the window.

Window

Window creates a window on the screen, to which output will be sent. (X1,Y1) are the coordinates of the upper left corner of the window, (X2,Y2) are the coordinates of the bottom right corner of the window. These coordinates are relative to the entire screen, with the top left corner equal to (1,1). Further coordinate operations, except for the next Window call, are relative to the window's top left corner.

```
Program Example;
uses Crt ;
{ Program to demonstrate the Window function }
begin
   ClrScr;
   WriteLn ('Creating a window from 30,10 to 50,20');
   Window(30,10,50,20);
   WriteLn ('We are now writing in this small window we just created');
   Write ('Press any key to clear the window');
   ReadKey ;
   ClrScr ;
   Write ('The window is cleared, press any key to restore to full screen');
   ReadKey ;
   {Full Screen is 80x25}
   Window(1,1,80,25);
   Clrscr ;
   Writeln ('Back in Full Screen');
end.
```

Réalisation d'un menu simple

```
program UMenuComplet;
uses crt;
var optionEncours : Integer;
   choix : Char;
begin
   optionEncours := 1; // C'est la premiere option qui est choisie au depart
   CursorOff;      // Rend le curseur invisible

   repeat
      TextBackground (Black);  // Remettre un fond noir
      ClrScr;  // Effacement de l'écran

      // Affichage d'un titre
      TextColor (LightRed);
      GotoXY(1,1);
      Writeln ('Titre de mon application');
      TextColor (White);

      // Affichage des options du menu
      GotoXY(1,3);
      if optionEncours = 1 then
         TextBackground (Green)
      else
         TextBackground (Black);
      Write ('Premiere option du menu');

      GotoXY(1,4);
      if optionEncours = 2 then
         TextBackground (Green)
      else
         TextBackground (Black);
      Write ('Deuxieme option du menu');

      GotoXY(1,5);
      if optionEncours = 3 then
         TextBackground (Green)
      else
         TextBackground (Black);
      Write ('Troisieme option du menu');

      GotoXY(1,6);
      if optionEncours = 4 then
         TextBackground (Green)
      else
         TextBackground (Black);
      Write ('Quatri' + Chr(138) + 'me option du menu');

      GotoXY(1,7);
      if optionEncours = 5 then
         TextBackground (Green)
      else
         TextBackground (Black);
      Write ('Quitter');
```

```
        // Test de la touche frappée
        choix := readkey;
        case choix of
          #0 : // Touche "spéciale"
            begin
              choix := readkey;   // Lecture du scancode
              case choix of
                #80 :
                  begin    // Down
                    if optionEncours = 5 then
                      optionEnCours := 1
                    else
                      optionEnCours := optionEncours + 1;
                  end;
                #72 :
                  begin    // Up
                    if optionEncours = 1 then
                      optionEnCours := 5
                    else
                      optionEnCours := optionEncours - 1;
                  end;
              end;
            end;   // Fin du cas #0

          #13 : // Enter
            begin
              Window(30,10,70,20); // Création d'une fenêtre
              TextColor (Blue);
              TextBackground (Yellow);
              Write('Vous avez choisi l''option ',optionEnCours);
              (*******************************************
               * C'est ici qu'on teste l'option choisie et *
               * qu'on exécute la bonne procédure          *
               *******************************************)
              Readln;
              Window(1,1,80,25);     // La fenêtre est tout l'écran
            end;   // Fin du cas #13
        end;   // Fin du Case choix
      until (choix = #13) and (optionEncours = 5);

  end.
```

Pour plus de détails concernant l'unité CRT, veuillez consulter le chapitre 40 du manuel de référence :

```
Free Pascal version 3.0.0:
Reference guide for RTL units.
Document version 3.0
November 2015
```

8.2 L'unité GRAPH

Cette unité contient des procédures et fonctions qui permettent de travailler au niveau du pixel donc de dessiner ou de réaliser des graphiques.

Cf. *The GRAPH unit* ou le chapitre 18 du manuel de référence mentionné ci-dessus.

8.3 L'unité WINCRT

> *If you use the **Graph** unit under Win32, you **can't use** the API mouse unit for mouse support or use the win32 **Crt unit to get keyboard data**. The reason for this is that the window popped up is a GUI window, and not a console one.*
>
> *The wincrt unit provides some auxiliary routines for **use with the graph unit**, namely keyboard support. It has no connection with the crt unit.*

8.3.1 Fonctions et procédures

delay	readkey
keypressed	sound

Cf. le chapitre 45 du manuel de référence mentionné ci-dessus.

Deuxième partie

Exercices supplémentaires

CHAPITRE 1

LES BOUCLES

1. Écrire un programme qui affiche les nombres de 1 à 10.
2. Écrire un programme qui affiche les nombres de 20 à 1 de trois en trois. Exemple :

   20 19 18
   17 16 15
   14 13 12
   ⋮

3. Écrire un programme qui calcule la somme des entiers de 1 à 100.
4. Même programme que précédemment, mais cette fois-ci l'utilisateur indique la limite. Par exemple, si l'on entre 10, le programme doit calculer :

   $$1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 = 55$$

5. Écrire un programme qui saisit un entier et qui l'affiche à l'envers. Par exemple, l'utilisateur saisit 123456 et le programme affiche 654321. Pour cela il faudra utiliser la division et le modulo.
   Rappel : 153 mod 10 = 3 et 153 div 10 = 15
   Remarque : Il est également possible d'utiliser les chaînes de caractères pour résoudre cet exercice.
6. Écrire un programme qui saisit deux nombres entiers positifs et calcule le premier à la puissance du second.
7. Écrire un programme qui demande un nombre de départ, et qui ensuite affiche les dix nombres suivants. Par exemple, si l'utilisateur entre le nombre 17, le programme affichera les nombres de 18 à 27.
8. Étant donné un entier naturel n, écrire un programme qui calcule la somme des carrés des nombres entiers inférieurs ou égaux à n. Pour 5, il faut calculer

   $$1 + 2^2 + 3^2 + 4^2 + 5^2 = 55$$

9. Écrire un programme qui saisit une date sous la forme de trois entiers (jour, mois et année). Le programme doit tester si la date est correcte, et si ce n'est pas le cas,

doit signaler le type d'erreur rencontrée, puis demander une nouvelle saisie. Le programme finit lorsqu'une date correcte est enfin saisie, avec l'affichage de celle-ci. Dans le cas où le mois de la date est février, votre programme devra calculer si l'année est bissextile [1]. De manière générale, il devra calculer le nombre de jours maximal du mois de la date saisie, de manière à valider le numéro de jour qui a été saisi.

10. Écrire un programme qui demande à l'utilisateur un jour, un mois et une année et qui affiche le lendemain.Par exemple, si l'utilisateur fournit :

    jour : 31
    mois : 12
    année : 1999

    le programme affiche : 1/01/2000.

11. Écrire un programme qui demande un nombre de départ, et qui ensuite écrit la table de multiplication de ce nombre, présentée comme suit (cas où l'utilisateur entre le nombre 7) :

    Table de 7 :
    7 x 1 = 7
    7 x 2 = 14
    7 x 3 = 21
    ...
    7 x 10 = 70

12. Écrire un programme qui demande un nombre de départ, et qui calcule la somme des inverses des entiers jusqu'à ce nombre. Par exemple, si l'on entre 5, le programme doit calculer :

    $$1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} = 2,2833$$

13. Écrire un programme qui demande un entier et qui calcule sa factorielle.
    NB : la factorielle de 8, notée $8!$, vaut $1 \times 2 \times 3 \times 4 \times 5 \times 6 \times 7 \times 8$ (= 40320)
14. Écrire un programme qui lit 10 nombres saisis au clavier et affiche le nombre de valeurs négatives lues.
15. Écrire un programme qui lit des nombres saisis au clavier et s'arrête au premier nombre négatif.
16. Écrire un programme qui demande successivement 8 nombres à l'utilisateur, et qui lui dise ensuite quel était le plus grand parmi ces 8 nombres.
17. Modifiez ensuite le programme pour qu'il affiche, en plus, en quelle position était le plus grand.
18. Réécrire le programme précédent, mais cette fois-ci on ne connaît pas d'avance combien l'utilisateur souhaite saisir de nombres. La saisie des nombres s'arrête lorsque l'utilisateur entre un zéro.
19. Le crible d'Ératosthène. La façon la plus simple de trouver des nombres premiers est d'utiliser un algorithme appelé crible d'Eratosthène (IIIe av. JC). L'idée est simple. Pour obtenir les nombres premiers inférieurs à n :

1. Une année A est bissextile si A est multiple de 4, toutefois, les années séculaires (siècles) ne sont bissextiles que si A est multiple de 400.

- Écrire tous les entiers de 2 à n,
- Enlever (ou barrer) les multiples de 2 sauf 2,
- Récupérer le plus petit nombre non barré, c'est à dire 3, et barrer les multiples de 3 sauf 3,
- Récupérer le plus petit nombre non barré, c'est à dire 5, et barrer les multiples de 5 sauf 5,
- ...
- On s'arrête dès qu'on a atteint $\sqrt{n}$.

Les nombres non barrés sont les nombres premiers.

20. L'utilisateur saisit un caractère, le programme teste s'il s'agit d'une lettre majuscule, si oui il renvoie cette lettre en minuscule, sinon il renvoie un message d'erreur.
21. Saisir une suite de caractères, compter et afficher le nombre de lettres 'e' et d'espaces.
22. L'utilisateur saisit le nom d'un fichier. Le programme vérifie que celui-ci possède l'extension .PAS
23. Écrire un programme qui remplace dans une chaîne de caractères les voyelles par des tirets ("-"). Par exemple, pour

    ```
    'la balle au bond rebondit bien',
    ```

    il faut obtenir

    ```
    'l- b-ll- -- b-nd r-b-nd-t b--n'
    ```

24. Écrire un programme qui calcule la valeur d'un nombre écrit en chiffres romains.
25. Écrire un programme qui affiche le nombre des entiers qui sont des multiples de 3 et inférieurs à un nombre n donné par l'utilisateur.
26. Écrire un programme qui affiche tous les entiers pairs inférieurs à une valeur entière n entrée par l'utilisateur.
27. On se propose de déterminer si un nombre est "couicable" (mot inventé), c'est-à-dire si la somme des chiffres de sa partie gauche est égale à la somme des chiffres de sa partie droite (le nombre de chiffres le composant doit être pair).
28. Écrire un programme qui lit un nombre entier et qui dessine ce qui suit (pour 10)

```
carre_plein(10)     carre_vide(10)     triangle(10)
**********          **********         *
**********          *        *         **
**********          *        *         ***
**********          *        *         ****
**********          *        *         *****
**********          *        *         ******
**********          *        *         *******
**********          *        *         ********
**********          *        *         *********
**********          **********         **********
```

29. Écrire un programme qui demande à l'utilisateur de taper 10 entiers et qui affiche la somme et le plus petit de ces entiers.

30. Écrire un programme affichant une table de multiplication sous la forme :

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

31. Écrire un programme qui demande à l'utilisateur de taper un entier n et qui calcule u_n défini par :

    $u_0 = 3$
    $u_{n+1} = 3u_n + 4$

32. Écrire un programme qui demande à l'utilisateur de taper un entier n et qui calcule u_n défini par :

    $u_0 = 1$
    $u_1 = 1$
    $u_{n+1} = u_n + u_{n-1}$

33. Écrire un programme qui permet de faire des opérations sur un entier (valeur initiale à 0). Le programme affiche la valeur de l'entier puis affiche le menu suivant :

    1 Ajouter 1
    2 Multiplier par 2
    3 Soustraire 4
    4 Quitter

    Le programme demande alors de taper un entier entre 1 et 4. Si l'utilisateur tape une valeur entre 1 et 3, on effectue l'opération, on affiche la nouvelle valeur de l'entier puis on réaffiche le menu et ainsi de suite jusqu'à ce qu'on tape 4. Lorsqu'on tape 4, le programme se termine.

34. Écrire un programme qui demande à l'utilisateur de taper des entiers strictement positifs et qui affiche leur moyenne. Lorsqu'on tape une valeur négative, le programme affiche *Erreur* et demande de retaper une valeur. Lorsqu'on tape 0, cela signifie que le dernier entier a été tapé. On affiche alors la moyenne. Si le nombre d'entiers tapés est égal à 0, on affiche *Pas de moyenne*.

35. Écrire un programme qui demande à l'utilisateur de saisir un entier n et qui affiche le nombre de nombres premiers inférieurs ou égaux à n.

36. Écrire un programme qui demande à l'utilisateur de saisir un entier N et qui affiche la figure suivante.

Pour N=1
*
Pour N=2
**
*
Pour N=3
***
**
*
⋮

37. Écrire un programme permettant d'estimer la valeur de π selon la formule suivante : $\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \cdots - \frac{1}{9999}$
38. Réduire une chaîne de caractères qui contient des répétitions : "aaaaaabbbccd" donne "abcd"
39. Extraire d'une chaîne de caractères différents éléments séparés entre eux par un caractère "séparateur". Par exemple, la chaîne "C :/Users/henrotte/Dropbox/Travaux/-Word" contient 6 éléments séparés par le caractère "/"
40. Extraire les n derniers caractères d'une chaîne
41. Effacer toutes les occurrences [2] d'une sous-chaîne
42. Remplacer toutes les occurrences d'une sous-chaîne par une autre
43. Rechercher où apparaît la n^{e} occurrence d'une chaîne
44. Le nombre 720 a 30 diviseurs (1, 2, 3, 4, 5, 6, 8, 9, ..., 360, 720). Un autre nombre, supérieur à 720 mais inférieur à 1000, en a encore plus. Écrire un programme qui permet de découvrir ce nombre et d'afficher ses diviseurs à raison de 5 diviseurs par ligne.
45. Écrire un programme permettant de trouver les 3 premiers couples de nombres *amicaux*. Informations complémentaires : deux nombres sont dits amicaux si la somme des diviseurs stricts de l'un est égale à l'autre.

    Exemple :
    diviseurs stricts de 220 : 1, 2, 4, 5, 10, 11, 20, 22, 44, 55, 110
    diviseurs stricts de 284 : 1, 2, 4, 71, 142
    220 et 284 sont deux nombres amicaux car :
    1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110 = 284
    1 + 2 + 4 + 71 + 142 = 220

    Solution : 220 et 284, 1184 et 1210, 2620 et 2924
46. La conjecture de Syracuse
    L'algorithme de Syracuse consiste à itérer l'opération suivante : à un nombre entier n, on associe $\frac{n}{2}$ si n est pair et $3n + 1$ si n est impair. On conjecture (on ne sait toujours pas si c'est vrai) que quel que soit l'entier considéré initialement dans cet algorithme, on arrive toujours à 1 après un certain nombre d'itérations. C'est en

2. Une *occurrence* est une apparition d'une unité linguistique dans un énoncé.

tout cas vrai pour tous les entiers avec lesquels l'algorithme a été testé. Écrire un programme qui demande à l'utilisateur un entier initial, qui effectue l'algorithme de Syracuse, affiche tous les nombres obtenus jusqu'au premier 1 et donne le nombre d'itérations effectuées jusqu'à l'obtention du premier 1.

47. Factorisation d'un entier
$2088 = 2^3 \times 3^2 \times 29$
Il s'agit tout simplement de diviser le nombre par 2 le plus possible, puis par 3 le plus possible, ...
Ainsi pour décomposer 2088 en produit de facteurs premiers

2088	2	2 divise 2088 le quotient est 1044
1044	2	2 divise 1044 le quotient est 522
522	2	2 divise 522 le quotient est 261
261	3	3 divise 261 le quotient est 87
87	3	3 divise 87 le quotient est 29
29		ni 3, ni 5 ne divisent 29 et 7^2 est plus grand que 29 (fin)

Autres exemples :
$$360 = 2^3 \times 3^2 \times 5$$
$$1050 = 2 \times 3 \times 5^2 \times 7$$
$$1827 = 3^2 \times 7 \times 29$$
$$4752 = 2^4 \times 3^3 \times 11$$

48. Le code de César consiste à crypter un message en remplaçant chaque lettre par celle qui se trouve 3 rangs à droite dans l'alphabet (et bien sûr "x", "y" et "z" deviennent respectivement "a", "b" et "c"). Par exemple "exemple" devient "hahpsoh". Écrire un algorithme qui crypte un mot entré au clavier en utilisant le code de César.

49. Combien un caissier a-t-il de façons de rendre la monnaie de n francs avec des pièces de 1, 2, 5 et 10 francs ?

50. Écrire un programme qui calcule le nombre *minimum* de billets et/ou de pièces qu'il faut pour un achat donné en euros sachant qu'il n'y a pas de centimes. Le programme doit indiquer combien et quels pièces et/ou billets ont été nécessaire.
    - Les billets sont 5, 10, 20, 50, 100, 200 et 500 €
    - Les pièces sont 1 et 2 €.

51. Un numéro de compte bancaire au format belge *BBAN* (*B*elgian *B*ank *A*ccount *N*umber) est formé de 12 chiffres (par exemple, 510-0075470-61). La vérification de la validité d'un numéro de compte se base sur le modulo 97 des 10 premiers chiffres, le résultat de l'opération donnant les deux derniers chiffres. Par exemple,
    $$5100075470 \bmod 97 = 61 \text{ (car } 5100075470 = 52578097 \times 97 + 61)$$
    Écrire un programme qui teste la validité d'un compte *BBAN*.

52. Écrire un programme qui calcule la racine carrée d'un réel a avec la suite récurrente :
    $$u_{n+1} = \tfrac{1}{2}(u_n + \tfrac{a}{u_n})$$
    avec une précision de 10^{-6}. Combien d'étapes a-t-il fallu ?

53. Écrire un programme demandant à l'utilisateur de saisir deux valeurs numériques x et p et affichant $\sqrt{x}$ avec une précision p. On utilisera une méthode par dichotomie. Par exemple, calculons la racine carrée de 10 avec une précision 0,001.
Dans l'exemple qui suit, m est le milieu de l'intervalle.
    - Commençons par la chercher dans $[0, 10]$, on a m = 5, comme $5^2 > 10$, alors $5 > \sqrt{10}$, donc $\sqrt{10}$ se trouve dans l'intervalle $[0, 5]$
    - On recommence, $m = 2.5$, comme $2.5^2 < 10$, alors $2.5 < \sqrt{10}$, on poursuit la recherche dans $[2.5, 5]$
    - On a $m = 3.75$, comme $3.75^2 > 10$, alors $3.75 > \sqrt{10}$, on poursuit la recherche dans $[2.5, 3.75]$
    - On a $m = 3.125$, comme $3.125^2 < 10$, alors $3.125 < \sqrt{10}$, on poursuit la recherche dans $[3.125, 3.75]$
    - $\cdots$
54. Déterminer le nombre m tel que $\sum_{k=1}^{m} \frac{1}{k} > n$; n étant un nombre choisi par l'utilisateur.
55. Écrire un programme qui calcule $\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+\cdots}}}}}$
56. Écrire un programme qui calcule $sin(x)$ en utilisant la formule d'approximation suivante :
$$sin(x) \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$
x et le nombre de termes seront lus au clavier.

57. Écrire un programme qui calcule une valeur approchée de π avec trois méthodes différentes.

Méthode 1

$$\frac{1}{2} - \frac{\pi}{8} \approx \sum_{k=1}^{n} \frac{1}{(4k-1)(4k+1)} = \frac{1}{3 \times 5} + \frac{1}{7 \times 9} + \frac{1}{11 \times 13} + \cdots$$

Si X est le résultat de l'approximation,

$$X = \frac{1}{2} - \frac{\pi}{8} \Longleftrightarrow \pi = 4 - 8X$$

Méthode 2

$$\frac{\pi^2}{6} \approx \sum_{k=1}^{n} \frac{1}{k^2} = 1 + \frac{1}{4} + \frac{1}{9} + \frac{1}{16} + \cdots$$

Si X est le résultat de l'approximation,

$$X = \frac{\pi^2}{6} \Longleftrightarrow \pi = \sqrt{6X}$$

Méthode 3

$$\frac{\pi^2}{8} \approx \sum_{k=0}^{n} \frac{1}{(2k+1)^2} = 1 + \frac{1}{9} + \frac{1}{25} + \frac{1}{49} + \cdots$$

Si X est le résultat de l'approximation,

$$X = \frac{\pi^2}{8} \Longleftrightarrow \pi = \sqrt{8X}$$

Le programme doit indiquer l'erreur commise avec chacune des méthodes pour n relativement grand ($n \geq 10000$).

58. Les nombres a tels que $(a+n+n^2)$ est premier pour tout n tel que $0 \leqslant n \leqslant (a-2)$ sont appelés nombres chanceux d'Euler.
5 est un nombre chanceux car $5+0+0=5$ est premier, $5+1+1=7$ est premier, $5+2+4=11$ est premier et $5+3+9=17$ est premier.
Leonhard Euler a identifié six nombres chanceux. Quels sont-ils ?.

59. Écrire un algorithme qui permette de connaître ses chances de gagner au tiercé, quarté, quinté. On demande à l'utilisateur le nombre de chevaux partants, et le nombre de chevaux joués. Les deux messages affichés devront être :

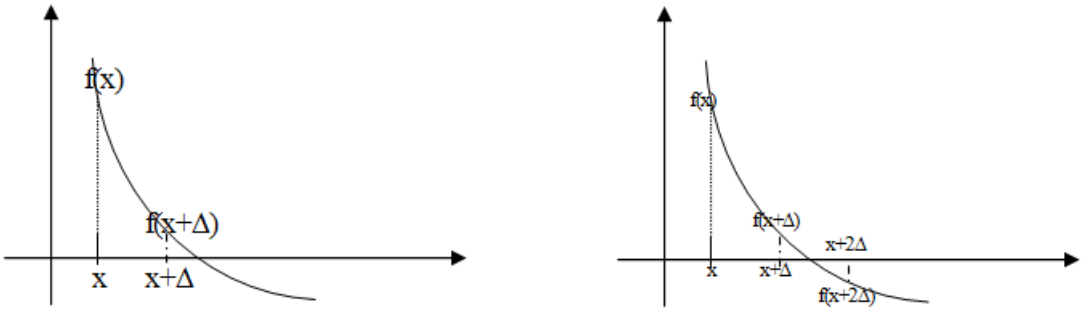
- Dans l'ordre : une chance sur X de gagner
- Dans le désordre : une chance sur Y de gagner

X et Y nous sont donnés par la formule suivante, si n est le nombre de chevaux partants et p le nombre de chevaux joués,

$$X = \frac{n!}{(n-p)!}$$

$$Y = \frac{n!}{p!(n-p)!}$$

60. Recherche d'une racine d'une fonction par la méthode des intervalles.
Commençant en un point arbitraire, f(x) est évalué en une série de points qui sont distants d'une valeur Δ donnée. Si on trouve que le signe de f(x) a changé entre deux évaluations successives, c'est qu'on a passé une racine.



(a) (b)

(a) Entre x *et* $x + \Delta$, la fonction f(x) ne change pas de signe

$$f(x) > 0 \text{ et } f(x + \Delta) > 0$$

c'est donc qu'il n'y a aucune racine entre x et $x + \Delta$.

(b) Entre $x + \Delta$ et $x + 2\Delta$, la fonction f(x) change de signe

$$f(x + \Delta) > 0 \text{ et } f(x + 2\Delta) < 0$$

c'est donc qu'il y a une racine entre $x + \Delta$ et $x + 2\Delta$. Dans ce cas, on recommence avec Δ diminué de moitié et dans l'autre sens (en réalité $\Delta = -\Delta/2$)

CHAPITRE 2

LES TABLEAUX

1. Générer un tableau de n éléments par des entiers aléatoires variant de 1 à n mais tous différents.
2. Écrire un programme qui indique le nombre de fois où un élément apparaît dans un tableau T d'entiers donnés.
3. Écrire un programme qui stocke la décomposition en facteurs premiers d'un nombre entier strictement positif dans un tableau et ensuite affiche les éléments de ce tableau sous la forme 18 = 2*3*3.
4. On dispose de 2 tableaux de même dimension *note* et *coef* qui contiennent respectivement les notes d'un étudiant (sur 20) et les coefficients attribués aux matières enseignées.
   note[i] est la note obtenue dans la matière *coef[i]*.
   Écrire un programme pascal qui permet de calculer la moyenne de l'élève.
5. Écrire un programme qui lit (ou génère) un tableau de 10 caractères et effectue les traitements suivants selon un menu :
   - calcule le nombre de voyelles,
   - compte le nombre d'occurrences d'un caractère,
   - convertit les caractères en majuscule,
   - affiche les caractères en horizontale,
   - affiche les caractères en verticale,
   - quitte.
6. Écrire un programme qui insère une valeur dans un tableau trié (en gardant bien sûr le tableau trié).
7. Étant donné un tableau T de n nombres positifs ou nuls, écrire un programme qui le tasse, c'est-à-dire qui détecte les éléments nuls du tableau et qui récupère leur place en décalant vers le début du tableau tous les autres éléments.
   Il ne faut donc utiliser qu'un seul tableau.
8. Écrire un programme qui calcule et renvoie la valeur d'un polynôme $P(X)$, représenté par le tableau de ses coefficients, pour une valeur donnée de la variable X.

9. Écrire un programme qui lit la dimension N d'un tableau T (dimension maximale : 50 composantes), remplit le tableau par des valeurs générées aléatoirement et affiche le tableau.
   Le programme copiera ensuite toutes les composantes strictement positives dans un deuxième tableau $TPOS$ et toutes les valeurs strictement négatives dans un troisième tableau $TNEG$. Afficher les tableaux $TPOS$ et $TNEG$.

10. Écrire un programme qui génère deux tableaux d'entiers et qui calcule le *schtroumph* [1] de ces deux tableaux. Pour le calculer, il faut multiplier chaque élément du premier tableau par chaque élément du second et additionner le tout. Par exemple,

    Le *schtroumph* de | 4 | 8 | 7 | 12 | et | 3 | 6 | vaut 279 car

    $$3 \times 4 + 3 \times 8 + 3 \times 7 + 3 \times 12 + 6 \times 4 + 6 \times 8 + 6 \times 7 + 6 \times 12 = 279$$

11. Soit T un tableau d'entiers. On suppose que ce tableau n'est pas trié. Écrire un programme qui affiche l'élément qui apparaît le plus souvent dans le tableau T, ainsi que son nombre d'occurrences. Si plusieurs éléments différents répondent au problème, votre programme doit en fournir un, quel qu'il soit.
    (a) l'utilisation d'un second tableau est permise,
    (b) il ne faut utiliser que le tableau T.

12. Soit T un tableau d'entiers de taille N. Une série dans T est une suite d'éléments consécutifs et égaux de T. Le problème consiste à trouver la plus longue série dans T, l'indice de son premier élément et sa longueur.
    Par exemple, pour

2	2	6	6	6	6	9	9	9	9	9	1	0	0	0

    le programme affichera la série est 9 elle commence à l'indice 7 et elle est de longueur 5.

13. Écrire un programme qui à partir d'un tableau d'entiers T, fournit le nombre de sous-séquences strictement croissantes de ce tableau, ainsi que les indices de début et de fin de la plus grande sous-séquence. Par exemple, si T vaut

    1 2 5 3 12 25 13 8 4 7 24 28 32 11 14

    Les séquences strictement croissantes sont :
    - 1, 2, 5
    - 3, 12, 25
    - 13
    - 8
    - 4, 7, 24, 28, 32
    - 11, 14.

    Le nombre de sous-séquences est : 6 et la plus grande sous-séquence est : <4, 7, 24, 28, 32>.

1. Mot inventé

14. Un point-clos est un élément d'une matrice qui est à la fois :
    - un *maximum* sur la ligne et
    - un *minimum* sur la colonne

    Exemples : $\begin{pmatrix} \underline{\mathbf{3}} & 2 & 1 \\ 6 & 5 & 4 \\ 9 & 8 & 7 \end{pmatrix}$ $\begin{pmatrix} 1 & 7 & 9 & 8 \\ 4 & \underline{\mathbf{6}} & 0 & \underline{\mathbf{6}} \end{pmatrix}$ $\begin{pmatrix} 0 & \underline{\mathbf{7}} \\ 8 & 9 \\ 6 & \underline{\mathbf{7}} \\ 8 & 8 \end{pmatrix}$

    Écrire un programme qui
    (a) permet de lire une matrice au clavier, à savoir,
        - lire le nombre de lignes,
        - lire le nombre de colonnes,
        - lire les différents éléments

    (b) affiche la matrice,

    (c) qui crée une nouvelle matrice $MaxLig$ à partir de la matrice saisie Mat tel que :

$$MaxLig[i,j] = \begin{cases} 1 \text{ si } Mat[i,j] \text{ est un maximum sur la ligne } i \\ 0 \text{ sinon} \end{cases}$$

    (d) qui crée une nouvelle matrice $MinCol$ à partir de la matrice saisie Mat tel que :

$$MinCol[i,j] = \begin{cases} 1 \text{ si } Mat[i,j] \text{ est un minimum sur la ligne } j \\ 0 \text{ sinon} \end{cases}$$

    (e) permet trouver et afficher tous les points-clos d'une matrice Mat (affiche les valeurs des éléments ainsi que leurs numéros de lignes et de colonnes) en utilisant les matrices $MaxLig$ et $MinCol$

15. Écrire un programme qui multiplie deux matrices.

$$\begin{pmatrix} 1 & 0 \\ -1 & 3 \end{pmatrix} \times \begin{pmatrix} 3 & 1 \\ 2 & 1 \end{pmatrix} = \begin{pmatrix} 3 & 1 \\ 3 & 2 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 & 0 \\ 4 & 3 & -1 \end{pmatrix} \times \begin{pmatrix} 5 & 1 \\ 2 & 3 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 9 & 7 \\ 23 & 9 \end{pmatrix}$$

$$\begin{pmatrix} 5 & 1 \\ 2 & 3 \\ 3 & 4 \end{pmatrix} \times \begin{pmatrix} 1 & 2 & 0 \\ 4 & 3 & -1 \end{pmatrix} = \begin{pmatrix} 9 & 13 & -1 \\ 14 & 13 & -3 \\ 19 & 18 & -4 \end{pmatrix}$$

Troisième partie

Correction de quelques exercices

CHAPITRE 1

LES STRUCTURE DE CONTRÔLE

1.1 Exercices élémentaires : séquence

1. Afficher bonjour à l'écran.

```
program Bonjour;
begin
    //Affiche Bonjour à l'écran, le curseur reste à la fin de Bonjour
    Write('Bonjour');

    Readln; //Pour éviter la fermeture de la fenêtre
end.
```

2. Afficher une carte de visite à l'écran.

```
program CarteDeVisite;
begin
    //Affichage de la carte de visite
    Writeln('Jean Tanghe'); // Writeln pour aller à la ligne après avoir écrit
    Writeln('rue des Alouettes, 68');
    Writeln('B-6760 Virton');

    Readln; //Pour éviter la fermeture de la fenêtre
end.
```

3. Calculer la somme et le produit de deux nombres réels.

```
program SommeProduit;
var nombre1, nombre2,       // Les deux nombres réels
   somme, produit : real;   // La somme et le produit
begin
    //Lecture des deux nombres réels
    Write('Quel est le premier nombre ? ');
    Readln(nombre1);
    Write('Quel est le second nombre ? ');
    Readln(nombre2);
```

```
    //Calcul de la somme et du produit
    somme := nombre1 + nombre2;
    produit := nombre1 * nombre2;

    //Affichage de la somme et du produit
    Writeln('La somme est ',somme:0:3); // :0 :3 pour afficher avec 3 décimales
    Writeln('Le produit est ',produit:0:3);

    Readln;
end.
```

4. Calculer l'aire et le périmètre d'un rectangle connaissant sa largeur et sa hauteur.

```
program AirePerimetre;
var   largeur, hauteur,          // La largeur et la hauteur
      aire, perimetre : real;    // L'aire et le périmètre
begin
    //Lecture de la largeur et de la hauteur
    Write('Quelle est la largeur du rectangle ? ');
    Readln(largeur);
    Write('Quelle est la hauteur du rectangle ? ');
    Readln(hauteur);

    //Calcul de l'aire et du périmètre
    aire := largeur * hauteur;
    perimetre := 2 * (largeur+hauteur);

    //Affichage de l'aire et du périmètre
    Writeln('L''aire est ',aire:0:3);  //deux ' pour en afficher une
    Writeln('Le perimetre est ',perimetre:0:3);

    Readln;
end.
```

5. Calculer le volume de la sphère.

$$V = \frac{4}{3}\pi R^3$$

```
program VolumeSphere;
var   rayon,      // Le rayon de la sphère
      volume : real; // Le volume de la sphère
begin
    // Lecture du rayon de la sphère
    Write('Quel est le rayon de la sphere ? ');
    Readln(rayon);

    //Calcul du volume de la sphère
    volume := 4/3*pi*rayon*rayon*rayon;

    //Affichage du volume de la sphère
    Writeln('Le volume de la sphere est ',volume:0:3);

    Readln;
end.
```

6. Échanger le contenu de deux variables.

```
program EchangeVariables;
var   variable1, variable2,   // Les deux variables
      echange              : string; // Une variable intermédiaire pour l'échange
            // On peut prendre un autre type (integer, real...)
begin
   // Lecture des deux variables
   Write('Quel est le contenu de la premiere variable ? ');
   Readln(variable1);
   Write('Quel est le contenu de la seconde variable ? ');
   Readln(variable2);

   // Échange des contenus
   echange := variable1; // On sauvegarde le contenu de variable1
   variable1 := variable2; // On place le contenu de variable2 dans variable1
   variable2 := echange; //On récupère le contenu de variable1 et on le place dans variable2

   // Affichage du contenu des variables
   Writeln; //Pour afficher une ligne blanche
   Writeln('Le contenu de la premiere variable est ',variable1);
   Writeln('Le contenu de la seconde variable est ',variable2);

   Readln;
end.
```

7. Calculer le quotient et le reste d'une division de deux entiers.

```
program QuotientReste;
var   dividende, diviseur,      // Le dividende et le diviseur
      quotient, reste  : integer;  // Le quotient et le reste
begin
   // Lecture du dividende et du diviseur
   Write('Quel est le dividende ? ');
   Readln(dividende);
   Write('Quel est le diviseur ? ');
   Readln(diviseur);

   // Calcul du quotient et du reste
   quotient := dividende div diviseur; // div pour la division entière
                          // Ne pas confondre avec / pour la division réelle
   reste:=dividende mod diviseur;//mod pour le reste d'une division entière (modulo)

   // Affichage du quotient et du reste
   Writeln;
   Writeln('Le quotient est ',quotient);
   Writeln('Le reste est ',reste);

   Readln;
end.
```

8. Ce programme lit une température en degrés Celcius et l'affiche en degrés Fahrenheit.

$$Farenheit = \frac{9}{5}Celsius + 32$$

```
program CelciusVersFahrenheit;
var   celcius,      // La température en degrés Celcius
      fahrenheit : real; // La température en degrés Fahtrenheit
begin
   // Lecture de la température en degrés Celcius
   Write('Quelle est la temperature en degres Celcius ? ');
   Readln(celcius);

   // Calcul de la température en degrés Fahrenheit
   fahrenheit := 9/5*celcius + 32;

   // Affichage de la température en degrés Fahrenheit
   Writeln('La temperature en degres Fahrenheit est ',fahrenheit:0:3);

   Readln;
end.
```

9. Ce programme lit une température en degrés Fahrenheit et l'affiche en degrés Celcius.

$$Celcius = \frac{5}{9}(Farenheit - 32)$$

```
program CelciusVersFahrenheit;
var   fahrenheit,    // La température en degrés Fahtrenheit
      celcius    : real; // La température en degrés Celcius
begin
   // Lecture de la température en degrés Celcius
   Write('Quelle est la temperature en degres Fahrenheit ? ');
   Readln(fahrenheit);

   // Calcul de la température en degrés Fahrenheit
   celcius := 5/9*(fahrenheit-32);

   // Affichage de la température en degrés Fahrenheit
   Writeln('La temperature en degres Celcius est ',celcius:0:3);

   Readln;
end.
```

10. Lire un nombre de secondes et le traduire en heures, minutes et secondes.

```
program SecondesHMS;
var   secondesTotales,          // Le nombre de secondes lues
      heures, minutes, secondes, // Le nombre d'heures, minutes et secondes
      reste           : integer;  // Le reste de la division
begin
   // Lecture du nombre de secondes
   Write('Quel est le nombre de secondes ? ');
   Readln(secondesTotales);

   // Calcul du nombre d'heures, minutes et secondes
   heures := secondesTotales div 3600;  //Il y a 3600 secondes dans une heure
   reste  := secondesTotales mod 3600; //Il reste encore des secondes
   minutes  := reste div 60; //Il y a 60 secondes dans une minute
   secondes := reste mod 60; //Il reste des secondes

   // Affichage des heures, minutes et secondes
   Writeln('Il y a ',heures,' heure(s) ',minutes,' minute(s) ',
         secondes,' seconde(s)');

   Readln;
end.
```

11. Ce programme transforme un angle donné en radians en degrés.

$$degre = \frac{180}{\pi} radian$$

```
program RadiansVersDegres;
var radians,degres : real; // L'angle en radians et en degrés
begin
   // Lecture de l'angle en radians
   Write('Quelle est la valeur de l''angle en radians ? ');
   Readln(radians);

   // Transformation de l'angle en degrés
   degres := 180/pi * radians;

   // Affichage de l'angle en degrés
   Writeln('L''angle mesure ',degres:0:3,' degre(s)');

   Readln;
end.
```

12. Ce programme transforme un angle donné en degrés en radians.

$$radian = \frac{\pi}{180} degre$$

```
program DegresVersRadians;
var degres,radians : real; //L'angle en degrés et en radians
begin
   // Lecture de l'angle en degrés
   Write('Quelle est la valeur de l''angle en degres ? ');
   Readln(degres);

   // Transformation de l'angle en radians
   radians := pi/180*degres;

   // Affichage de l'angle en radians
   Writeln('L''angle mesure ',radians:0:3,' radian(s)');
   Readln;
end.
```

13. Ce programme transforme un angle donné en degrés, minutes, secondes en radians.

```
program DMSVersRadians;
var    degres,minutes,secondes:integer; // L'angle en degrés, minutes et secondes
       degresDecimaux,radians : real; // L'angle en degrés et en radians
begin
   // Lecture de l'angle en degrés, minutes et secondes
   Write('Quel le nombre de degres ? ');
   Readln(degres);
   Write('Quel le nombre de minutes ? ');
   Readln(minutes);
   Write('Quel le nombre de secondes ? ');
   Readln(secondes);

   // Transformation de l'angle en radians
   degresDecimaux := degres + minutes/60 + secondes/3600;
   radians := pi/180 * degresDecimaux;

   // Affichage de l'angle en radians
   Writeln('L''angle mesure ',radians:0:3,' radian(s)');

   Readln;
end.
```

14. Ce programme transforme un angle donné en radians en degrés, minutes, secondes et millièmes de secondes.

```
program RadiansVersDMSm;
var   degres, minutes, secondes, milliemes : integer;
      // L'angle en degrés, minutes, secondes, millièmes de seconde
   degresDecimaux,minutesDecimales,secondesDecimales,radians : real;
      // L'angle en degrés, minutes et en secondes décimales, et l'angle en radians
begin
   // Lecture de l'angle en radians
   Write('Quel le nombre de radians ? ');
   Readln(radians);

   // Transformation de l'angle en degrés, minutes, secondes et millièmes de seconde
   degresDecimaux := 180/pi * radians;
   degres := trunc(degresDecimaux); // Trunc = partie entière
   minutesDecimales := frac(degresDecimaux) * 60; //Frac = partie décimale
   minutes := trunc(minutesDecimales);
   secondesDecimales := frac(minutesDecimales) * 60;
   secondes := trunc(secondesDecimales);
   milliemes := round(frac(secondesDecimales) * 1000); //Round = arrondi

   // Affichage de l'angle en degrés, minutes, secondes et millièmes de seconde
   Writeln('L''angle mesure ',degres,' degre(s) ',minutes,' minute(s) ',
   secondes,' seconde(s) ',milliemes,' millieme(s)');

   Readln;
end.
```

15. Ce programme lit un nombre de jours et le transforme en années, mois, jours en supposant qu'un mois contiendra toujours 30 jours.

```
program JoursVersAnneesMoisJours;
var joursTotal,annees,mois,jours:integer; // Le nombre de jours total, d'années,
        //de mois et de jours
   reste : integer; //Reste de division (pour le calcul)
begin
   // Lecture du nombre de jours total
   Write('Quel est le nombre de jours ? ');
   Readln(joursTotal);

   // Transformation du nombre de jours en années, mois et jours
   annees := joursTotal div 360;
   reste  := joursTotal mod 360;
   mois := reste div 30;
   jours  := reste mod 30;

   // Affichage du nombre d'années, mois et jours
   Writeln;
   Writeln('Il y a ',annees,' annee(s) ',mois,' mois ',jours,' jour(s)');

   Readln;
end.
```

16. Calculer la résistance totale de trois résistances en parallèle.

$$\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3}$$

```
program ResistanceParallele;
var R1,R2,R3,R : real; // Les trois résistances et la résistance totale
begin
   // Lecture des trois résistances
   Write('Quelle est la valeur de la premiere resistance ? ');
   Readln(R1);
   Write('Quelle est la valeur de la deuxieme resistance ? ');
   Readln(R2);
   Write('Quelle est la valeur de la troisieme resistance ? ');
   Readln(R3);

   // Calcul de la résistance totale
   R := 1/R1 + 1/R2 + 1/R3;
   R := 1/R;

   // Affichage de la résistance totale
   Writeln('La resistance totale est ',R:0:3,' ohm(s)');

   Readln;
end.
```

17. Calculer le périmètre et l'aire d'un triangle dont on connaît la longueur des 3 côtés. Si p est le demi-périmètre ($p = \frac{a+b+c}{2}$), alors

$$Aire = \sqrt{p(p-a)(p-b)(p-c)}$$

```
program AirePerimetreTriangle;
var   cote1, cote2, cote3,          // Les trois cotés
      perimetre, aire     : real;   // Le périmètre et l'aire
begin
   // Lecture des côtés
   Write('Quelle est la longueur du premier cote ? ');
   Readln(cote1);
   Write('Quelle est la longueur du deuxieme cote ? ');
   Readln(cote2);
   Write('Quelle est la longueur du troisieme cote ? ');
   Readln(cote3);

   // Calcul du périmètre et de l'aire
   perimetre := cote1 + cote2 + cote3;
   aire  := sqrt  (perimetre/2 * (perimetre/2-cote1) *
                   (perimetre/2-cote2) * (perimetre/2-cote3));

   // Affichage du périmètre et de l'aire
   Writeln('Le perimetre est ',perimetre:0:3);
   Writeln('L''aire est ',aire:0:3);

   Readln;
end.
```

18. Ce programme lit trois chaînes et les affiche en une seule.

```
program TroisChaines;
var chaine1,chaine2,chaine3,chaineTotale:string;
      // Les trois chaînes et la chaîne totale
begin
   / /Lecture des trois îchanes
   Write('Quelle est la premiere chaine ? ');
   Readln(chaine1);
   Write('Quelle est la deuxieme chaine ? ');
   Readln(chaine2);
   Write('Quelle est la troisieme chaine ? ');
   Readln(chaine3);

   // Calcul de la chaine totale
   chaineTotale := chaine1 + chaine2 + chaine3;

   // Affichage de la chaîne totale
   Writeln('La chaine est ',chaineTotale);

   Readln;
end.
```

19. Calculer le temps écoulé entre deux événements.
Le plus simple est d'obtenir l'heure juste avant (**temps1**) et juste après (**temps2**) l'événement et d'utiliser la formule :

$$(temps2 - temps1) * 24 * 3600$$

L'heure courante est donnée par la fonction *Time*.

1.2 Les tests

1. Afficher le plus grand et le plus petit de deux nombres.

```
program GrandPetit;
var nombre1, nombre2 : real; // Les deux nombres
begin
   // Lecture des deux nombres
   Write('Quel est le premier nombre ? ');
   Readln(nombre1);
   Write('Quel est le second nombre ? ');
   Readln(nombre2);

   // Calcul est affichage du plus grand et du plus petit
   if nombre1 > nombre2 then
   begin
      Writeln('Le plus grand nombre est ',nombre1:0:3,
              ' et le plus petit est ',nombre2:0:3);
   end
   else
   begin
      Writeln('Le plus grand nombre est ',nombre2:0:3,
              ' et le plus petit est ',nombre1:0:3);
   end;

   Readln;
end.
```

2. Ce programme calcule la racine carrée d'un nombre si elle existe.

```
program RacineCarree;
var nombre, racine : real; // Le nombre et la racine
begin
   //Lecture du nombre
   Write('Quel est le nombre ? ');
   Readln(nombre);

   //Calcul et affichage de la racine si elle existe
   if nombre >= 0 then
   begin
      racine := sqrt(nombre);
      Writeln('La racine est ',racine:0:3);
   end
   else
   begin
      Writeln('La racine d''un nombre negatif n''existe pas !');
   end;

   readln;
end.
```

3. Résoudre l'équation du second degré $ax^2 + bx + c = 0$

```
program EquationDuSecondDegre;
var a,b,c,      // Les coefficients de l'équation du second degré
    delta,      // Delta = b^2 - 4ac
    x1,x2   : real; // Les éventuelles racines
begin
    // Lecture des coefficients de l'équation
    Writeln('Resolution de l''equation du second degre');
    Writeln;
    Write('Quelle est la valeur de a ? ');
    Readln(a);
    Write('Quelle est la valeur de b ? ');
    Readln(b);
    Write('Quelle est la valeur de c ? ');
    Readln(c);
    Writeln;

    // Calcul et affichage des racines
    delta := b*b - 4*a*c;

    if delta > 0 then
    begin
        x1 := (-b-sqrt(delta))/(2*a);
        x2 := (-b+sqrt(delta))/(2*a);
        Writeln('Les racines sont ',x1:0:3,' et ',x2:0:3);
    end
    else
    begin
        if delta = 0 then
        begin
            x1 := -b/(2*a);
            Writeln('La racine double est ',x1:0:3);
        end
        else
        begin
            Writeln('Il n''y a pas de racine !');
        end;
    end;
    Readln;
end.
```

4. Calculer le plus grand et le plus petit parmi trois nombres.

```
program PlusGrandPlusPetit;
var nombre1,nombre2,nombre3:Real; // Les trois nombres
begin
   // Lecture des 3 nombres
   Write('Quel est le premier nombre ? '); Readln(nombre1);
   Write('Quel est le deuxieme nombre ? '); Readln(nombre2);
   Write('Quel est le troisieme nombre ? '); Readln(nombre3);

   // Calcul et affichage du plus petit
   if (nombre1 <= nombre2) and (nombre1 <= nombre3) then
      writeln (nombre1:0:3, ' est le plus petit')
   else
      if (nombre2 <= nombre1) and (nombre2 <= nombre3) then
         writeln (nombre2:0:3, ' est le plus petit')
      else
         writeln (nombre2:0:3, ' est le plus petit');

   // Calcul et affichage du plus grand
   ⋮

   Readln;
end.
```

Une autre solution

```
program PlusGrandPlusPetit;
var nombre1,nombre2,nombre3:Real; //Les trois nombres
begin
   //Lecture des 3 nombres
   Write('Quel est le premier nombre ? '); Readln(nombre1);
   Write('Quel est le deuxieme nombre ? '); Readln(nombre2);
   Write('Quel est le troisieme nombre ? '); Readln(nombre3);

   //Calcul et affichage du plus petit
   if (nombre1 <= nombre2) and (nombre1 <= nombre3) then
      writeln (nombre1:0:3, ' est le plus petit');
   if (nombre2 <= nombre1) and (nombre2 <= nombre3) then
      writeln (nombre2:0:3, ' est le plus petit')
   if (nombre3 <= nombre1) and (nombre3 <= nombre2) then
      writeln (nombre3:0:3, ' est le plus petit');

   // Calcul et affichage du plus grand
   ⋮

   Readln;
end.
```

5. Calculer le plus grand et le plus petit parmi *six* nombres.

```
program PlusGrandPlusPetit;
var a,b,c,d,e,f : real; // Les 6 nombres
    min,max : real; // Le plus petit et le plus grand
begin
    // Lecture des 6 nombres
    Write('Quel est le premier nombre ? ');
    Readln(a);
    Write('Quel est le deuxieme nombre ? ');
    Readln(b);
    Write('Quel est le troisieme nombre ? ');
    Readln(c);
    Write('Quel est le quatrieme nombre ? ');
    Readln(d);
    Write('Quel est le cinquieme nombre ? ');
    Readln(e);
    Write('Quel est le sixieme nombre ? ');
    Readln(f);

    // Calcul du plus petit
    min := a;
    if b < min then
       min := b;
    if c < min then
       min := c;
    if d < min then
       min := d;
    if e < min then
       min := e;
    if f < min then
       min := f;

    // Calcul du plus grand
    max := a;
    if b > max then
       max := b;
    if c > max then
       max := c;
    if d > max then
       max := d;
    if e > max then
       max := e;
    if f > max then
       max := f;

    // Affichage du plus petit et du plus grand
    Writeln;
    Writeln(min:0:3,' est le plus petit !');
    Writeln(max:0:3,' est le plus grand !');

    Readln;
end.
```

6. Déterminer si une année est bissextile (une année est bissextile si elle est multiple de 4. Toutefois les années séculaires ne sont bissextiles que si elles sont multiple de 400).

```
program AnneeBissextile;
var annee:integer; //L'année
begin
    //Lecture de l'année
    Write('Quelle est l''annee ? ');
    Readln(annee);

    //Détermination du caractère de l'année (bissextile ou non)
    if annee mod 400 = 0 then
    begin
        Writeln('L''annee est bissextile !');
    end
    else
    begin
        if annee mod 100 = 0 then
        begin
            Writeln('L''annee n''est pas bissextile !');
        end
        else
        begin
            if annee mod 4 = 0 then
            begin
                Writeln('L''annee est bissextile !');
            end
            else
            begin
                Writeln('L''annee n''est pas bissextile !');
            end;
        end;
    end;
    Readln;
end.
```

7. Déterminer si un nombre de trois chiffres est d'Armstrong (un nombre est d'Armstrong si la somme des cubes de ses trois chiffres est égale au nombre).

```
program Armstrong;
var nombre:integer;   // Le nombre de trois chiffres
    u,d,c:integer; // Le chiffre des unités, des dizaines et des centaines
begin
    // Lecture du nombre
    Write('Quel est le nombre ? ');
    Readln(nombre);

    // Décomposition du nombre en trois chiffres
    u := nombre mod 10;
    d := nombre div 10 mod 10;
    c := nombre div 100;
```

```
    // On regarde si le nombre est d'Armstrong
    if u*u*u + d*d*d + c*c*c = nombre then
    begin
        Writeln(nombre,' est un nombre d''Armstrong !');
    end
    else
    begin
        Writeln(nombre,' n''est pas un nombre d''Armstrong !');
    end;

    Readln;
end.
```

8. Déterminer si un triangle est rectangle en connaissant ses 3 côtés.

```
program TriangleRectangle;
var a,b,c : real; // Les trois côtés
begin
    // Lecture des trois côtés
    Write('Quelle est la longueur du premier cote ? ');
    Readln(a);
    Write('Quelle est la longueur du deuxieme cote ? ');
    Readln(b);
    Write('Quelle est la longueur du troisieme cote ? ');
    Readln(c);

    //On vérifie si le triangle est rectangle en vérifiant Pythagore
    if (a*a = b*b + c*c) or (b*b = a*a + c*c) or (c*c = a*a + b*b) then
    begin
        Writeln('Le triangle est rectangle !');
    end
    else
    begin
        Writeln('Le triangle n''est pas rectangle !');
    end;

    Readln;
end.
```

9. Ce programme calcule le prix à payer si une réduction de 5% est accordée sur la partie qui dépasse 100 €.

```
program Reduction100;
var    montantInitial, reduction : real; //Le montant initial, la réduction
    montantAPayer : real;       // et le trotal à payer
begin
   // Lecture du montant initial
   Write('Quel est le montant total de vos achats ? ');
   Readln(montantInitial);

   // Calcul du montant à payer
   if montantInitial < 100 then
   begin
      reduction := 0;
   end
   else
   begin
      reduction := (montantInitial-100) * 5/100;
   end;
   montantAPayer := montantInitial - reduction;

   // Affichage du montant à payer
   Writeln('Vous devez payer ',montantAPayer:0:2,' euro(s)');

   Readln;
end.
```

10. Ce programme affiche la position de la droite $ax + by + c = 0$ par rapport aux axes

```
program Droite;
var a,b,c : real; // Les coefficients de la droite
begin
   // Lecture des coefficients de la droite
   Writeln('Position de la droite ax+by+c=0');
   Writeln;
   Write('Quelle est la valeur du coefficient "a" ? ');
   Readln(a);
   Write('Quelle est la valeur du coefficient "b" ? ');
   Readln(b);
   Write('Quelle est la valeur du coefficient "c" ? ');
   Readln(c);
```

```
    // Détermination et affichage de la position de la droite par rapport aux axes
    if a <> 0 then
    begin
        if b <> 0 then
        begin
            Writeln('La droite est oblique !');
        end
        else
        begin
            Writeln('La droite est parallele a Oy !');
        end;
    end
    else
    begin // a=0
        if b <> 0 then
        begin
            Writeln('La droite est parallele a Ox !');
        end
        else  // a=0 et b=0
        begin
            Writeln('Il n''y a pas de droite !');
        end;
    end;

    Readln;
end.
```

11. Lire un caractère, le programme teste s'il s'agit d'une lettre majuscule, si oui, il renvoie cette lettre en minuscule, sinon il renvoie un message d'erreur.

```
program TesteMajuscule;
var c : char;
begin
    // Lecture du caractère
    write('Quel est le caractere ? ');
    readln(c);

    // Teste si majuscule
    if (c >= 'A') and (c <= 'Z')  then
    begin
        c := LowerCase(c);
        writeln (c);
    end
    else
        writeln ('Erreur');

    readln;
end.
```

12. Lire deux chaînes de caractères et afficher à partir de quelle position commence la première dans la seconde.

```
program Cherche;
var chaine1,chaine2:string;    // Les deux chaînes
      endroit: integer; // Endroit où se trouve chaine2 dans chaine1
begin
   //Lecture des deux chaines
   Write('Quelle est la premiere chaine ? ');
   Readln(chaine1);
   Write('Quelle est la deuxieme chaine ? ');
   Readln(chaine2);

   // Recherche
   endroit := Pos(chaine2,chaine1);

   // Affichage
   if endroit > 0 then
      writeln(chaine2,' se trouve en ',endroit,' position dans ', chaine1)
   else
      writeln(chaine2,' ne se trouve pas dans ', chaine1);
   readln;
end.
```

13. Lire un caractère et déterminer si ce caractère est une lettre, un chiffre ou un autre caractère.

```
program CaractereLettreChiffreAutre;
const lettres = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'; // Toutes les lettres
      chiffres = '0123456789'; // Tous les chiffres
var caractere : char; // Le caractère lu
begin
   //Lecture du caractère
   Write('Quel est le caractere ? ');
   Readln(caractere);

   // Détermination du caractère (lettre, chiffre ou autre)
   if pos(upcase(caractere),lettres) > 0 then
   begin
      Writeln('Le caractere est une lettre !');
   end
   else
   begin
      if pos(upcase(caractere),chiffres) > 0 then
      begin
         Writeln('Le caractere est un chiffre !');
      end
      else
      begin
         Writeln('Le caractere n''est ni une lettre, ni un chiffre !');
      end;
   end;

   Readln;
end.
```

14. Lire une chaîne et déterminer si elle commence ou se termine par une consonne ou une voyelle.

```
program ChaineConsonneVoyelle;
const consonnes='BCDFGHJKLMNPQRSTVWXZ'; // Toutes les consonnes
      voyelles='AEIOUY'; // Toutes les voyelles
var chaine:string; //La chaîne
begin
   //Lecture de la chaîne
   Write('Quelle est la chaine ? ');
   Readln(chaine);

   //On vérifie si le premier caractère est une consonne ou une voyelle
   if pos(upcase(chaine[1]),consonnes) > 0 then
   begin
      Writeln('Le premier caractere est une consonne !');
   end
   else
   begin
      if pos(upcase(chaine[1]),voyelles) > 0 then
      begin
         Writeln('Le premier caractere est une voyelle !');
      end
      else
      begin
         Writeln('Le premier caractere n"est ni une consonne, ni une voyelle!');
      end;
   end;

   //On vérifie si le dernier caractère est une consonne ou une voyelle
   if pos(upcase(chaine[length(chaine)]),consonnes) > 0 then
   begin
      Writeln('Le dernier caractere est une consonne !');
   end
   else
   begin
      if pos(upcase(chaine[length(chaine)]),voyelles) > 0 then
      begin
         Writeln('Le dernier caractere est une voyelle !');
      end
      else
      begin
         Writeln('Le dernier caractere n"est ni une consonne, ni une voyelle!');
      end;
   end;

   Readln;
end.
```

15. Ce programme lit deux réels et un symbole au clavier et
    - additionne les deux réels si le symbole est "+"
    - soustrait le second du premier si le symbole est "-"
    - multiplie les deux réels si le symbole est "*" ou "."
    - divise le premier par le second si le symbole est "/" ou " :"
    - affiche une erreur si le symbole ne vaut aucune de ces valeurs

```
program Calculatrice;
var nombre1,nombre2 : real; // Les deux nombres
    operateur : char; // L'opérateur
    resultat : real;  // Résultat de l'opération
begin
    // Lecture des deux nombres et de l'opérateur
    Write('Quel est le premier nombre ? ');
    Readln(nombre1);
    Write('Quel est le deuxieme nombre ? ');
    Readln(nombre2);
    Write('Quel est l''operateur ? ');
    Readln(operateur);

    // Calcul et affichage du résultat
    case operateur of
        '+': begin
                resultat:=nombre1+nombre2;
                Writeln('La somme est ',resultat:0:3);
             end;
        '-': begin
                resultat:=nombre1-nombre2;
                Writeln('La difference est ',resultat:0:3);
             end;
        '*','.' : begin
                      resultat:=nombre1*nombre2;
                      Writeln('Le produit est ',resultat:0:3);
                  end;
        '/',':' : begin
                      if nombre2<>0 then
                      begin
                          resultat:=nombre1/nombre2;
                          Writeln('Le quotient est ',resultat:0:3);
                      end
                      else
                      begin
                          Writeln('Division par zero !');
                      end;
                  end;
        else  // Du Case Of
            Writeln('L''operateur n''est pas valide !');
    end;

    Readln;
end.
```

16. Ce programme lit deux chaînes de caractères et supprime la seconde de la première si elle s'y trouve.

```
program Supprimerchaine;
var chaine1, chaine2 : string;
begin
   // Lecture des deux chaînes
   Write('Quelle est la chaine ? ');
   Readln(chaine1);
   Write('Quelle est la partie a supprimer ? ');
   Readln(chaine2);

   // Vérification de la présence de la seconde chaine dans la première et suppression
   if pos(chaine2,chaine1) > 0 then
   begin
      delete(chaine1,pos(chaine2,chaine1),length(chaine2));//Suppression
      Writeln('La chaine devient ',chaine1);
   end
   else
   begin
      Writeln('La seconde chaine n''est pas dans la premiere !');
   end;

   Readln;
end.
```

17. Lire le nom d'un fichier (une chaîne). Le programme vérifie que celui-ci possède l'extension « PAS ».

```
program Exercice15;
var fichier, extension : string;
   position : integer;
begin
   write ('Quel est le nom du fichier ? ');
   readln (fichier);

   position := pos('.',fichier);
   extension := copy(fichier,position + 1,length(fichier)-position);

   if position = 0 then
      writeln ('Ce n''est pas un nom valable.')
   else
   begin
      if extension = 'pas' then
         writeln ('C''est un fichier PAS.')
      else
         writeln ('C''est un fichier ',extension);
   end;

   readln;
end.
```

18. Lire une expression sous forme de chaîne et l'évaluer.

Simplification On suppose qu'on évalue une expression simple, c'est-à-dire deux opérandes entières et un opérateur.

```
program EvalSimple;
uses sysUtils; // pour Trim
var expression : String;
    strOperande1, strOperande2 : string; // Opérandes sous forme de chaîne
    operande1, operande2 : integer;      // Opérandes
    code : integer; // Pour Val
    ouEstPlus, ouEstMoins, ouEstFois, ouEstDivise,
       ouEstDiv, ouEstMod : integer; // position de l'opérateur
begin
    // Lecture de l'expression
    write('Quelle est l''expression ? ');
    readln (expression);

    // Recherche la position de l'opérateur
    ouEstPlus   := Pos('+',expression);
    ouEstMoins  := Pos('-',expression);
    ouEstFois   := Pos('*',expression);
    ouEstDivise := Pos('/',expression);
    ouEstDiv    := Pos('div',expression);
    ouEstMod    := Pos('mod',expression);

    // Calculer
    if ouEstPlus > 0 then
    begin
       strOperande1 := Copy(expression,1,ouEstPlus-1);
       strOperande2 :=
          Copy(expression,ouEstPlus+1,Length(expression)-ouEstPlus);
       strOperande1 := Trim(strOperande1);
       strOperande2 := Trim(strOperande2);
       Val(strOperande1,operande1,code);
       Val(strOperande2,operande2,code);
       writeln(operande1, ' + ', operande2, ' = ', operande1 + operande2);
    end
    else
    if ouEstMoins > 0 then
    begin
       strOperande1 := Copy(expression,1,ouEstMoins-1);
       strOperande2 :=
          Copy(expression,ouEstMoins+1,Length(expression)-ouEstMoins);
       strOperande1 := Trim(strOperande1);
       strOperande2 := Trim(strOperande2);
       Val(strOperande1,operande1,code);
       Val(strOperande2,operande2,code);
       writeln(operande1, ' - ', operande2, ' = ', operande1 - operande2);
    end
```

```
    else
    if ouEstFois > 0 then
    begin
       strOperande1 := Copy(expression,1,ouEstFois-1);
       strOperande2 :=
          Copy(expression,ouEstFois+1,Length(expression)-ouEstFois);
       strOperande1 := Trim(strOperande1);
       strOperande2 := Trim(strOperande2);
       Val(strOperande1,operande1,code);
       Val(strOperande2,operande2,code);
       writeln(operande1, ' * ', operande2, ' = ', operande1 * operande2);
    end
    else
    if ouEstDivise > 0 then
    begin
       strOperande1 := Copy(expression,1,ouEstDivise-1);
       strOperande2 :=
          Copy(expression,ouEstDivise+1,Length(expression)-ouEstDivise);
       strOperande1 := Trim(strOperande1);
       strOperande2 := Trim(strOperande2);
       Val(strOperande1,operande1,code);
       Val(strOperande2,operande2,code);
       writeln(operande1,' / ',operande2,' = ',(operande1 / operande2):0:2);
    end
    else
    if ouEstDiv > 0 then
    begin
       strOperande1 := Copy(expression,1,ouEstDiv-1);
       strOperande2 :=
          Copy(expression,ouEstDiv+4,Length(expression)-ouEstDiv-3);
       strOperande1 := Trim(strOperande1);
       strOperande2 := Trim(strOperande2);
       Val(strOperande1,operande1,code);
       Val(strOperande2,operande2,code);
       writeln(operande1,' div ',operande2,' = ',operande1 div operande2);
    end
    else
    if ouEstMod > 0 then
    begin
       strOperande1 := Copy(expression,1,ouEstMod-1);
       strOperande2 :=
          Copy(expression,ouEstMod+4,Length(expression)-ouEstMod-3);
       strOperande1 := Trim(strOperande1);
       strOperande2 := Trim(strOperande2);
       Val(strOperande1,operande1,code);
       Val(strOperande2,operande2,code);
       writeln(operande1,' mod ',operande2,' = ',operande1 mod operande2);
    end
    else
       writeln('éOprateur manquant!');

    readln;
end.
```

1.3 Les boucles

Passage d'une boucle à l'autre

$I \longleftarrow 1$
tant que $I \leq N$ **faire**
  instruction(s)
  $I \longleftarrow I + 1$
fin

$I \longleftarrow 1$
répéter
  instruction(s)
  $I \longleftarrow I + 1$
jusqu'à $I >$ N

pour $I \leftarrow 1$ **à** N **faire**
  instruction(s)
fin

1. Calculer la somme et le produits des n premiers entiers.
   (a) Somme

```
program SommeNPremiersEntiers;
var     n,                      // Le nombre d'entiers
        nombre,                 // Le nombre ajouté
        somme   : integer;      // La somme
begin
    //Lecture du nombre d'entiers
    Write('Combien d''entiers voulez-vous dans la somme ? ');
    Readln(n);

    //Calcul de la somme avec WHILE DO
    somme  := 0;   // Au départ, la somme vaut 0
    nombre := 1;   // Le premier terme de la somme est 1
    while nombre <= n do   // Le dernier terme est n
    begin
        somme := somme + nombre;   // On ajoute le nombre à la somme
        nombre:= nombre + 1;       // On passe au nombre suivant
    end;

    // Affichage de la somme
    Writeln('Avec la boucle WHILE, la somme est ',somme);

    // Calcul de la somme avec REPEAT UNTIL
    somme := 0;   // Au départ, la somme vaut 0
    nombre:= 1;   // Le premier terme de la somme est 1
    repeat
        somme := somme + nombre;   // On ajoute le nombre à la somme
        nombre:= nombre + 1;       // On passe au nombre suivant
    until nombre > n;   // Le dernier terme est n

    //Affichage de la somme
    Writeln('Avec la boucle REPEAT, la somme est ',somme);

    //Calcul de la somme avec FOR TO
    somme := 0;   // Au départ, la somme vaut 0
    for nombre := 1 to n do   //Le premier terme est 1 et le dernier est n
    begin
        somme := somme + nombre;   //On ajoute le nombre à la somme
    end;   //nombre est incrémenté automatiquement

    //Affichage de la somme
    Writeln('Avec la boucle FOR, la somme est ',somme);

    Readln;
end.
```

(b) Produit

```
program factorielle;
var n,           // Le nombre d'entiers
    nombre,      // Le nombre à multiplier
    produit   : integer; // Le produit
begin
    // Lecture du nombre d'entiers
    Write('Combien d''entiers voulez-vous dans le produit ? ');
    Readln(n);

    // Calcul du produit avec WHILE DO
    produit := 1;  // Au départ, le produit vaut 1
    nombre  := 1; // Le premier facteur est 1
    while nombre <= n do  // Le dernier facteur est n
    begin
        produit := produit * nombre;  // On multiplie le produit par le nombre
        nombre  := nombre + 1;  // On passe au nombre suivant
    end;
    // Affichage du produit
    Writeln('Avec la boucle WHILE, le produit est ',produit);

    //Calcul de la somme avec REPEAT UNTIL
    produit := 1;  //Au départ, le produit vaut 1
    nombre  := 1; //Le premier facteur est 1
    repeat
        produit := produit * nombre;  // On multiplie le produit par le nombre
        nombre := nombre + 1;  // On passe au nombre suivant
    until nombre > n;  // Le dernier terme est n
    // Affichage du produit
    Writeln('Avec la boucle REPEAT, le produit est ',produit);

    // Calcul de la somme avec FOR TO
    produit := 1;  //Au départ, le produit vaut 1
    for nombre := 1 to n do  //Le premier facteur est 1 et le dernier est n
    begin
        produit:=produit * nombre;  //On multiplie le produit par le nombre
    end; //nombre est incrémenté automatiquement
    // Affichage du produit
    Writeln('Avec la boucle FOR, le produit est ',produit);

    Readln;
end.
```

2. Lire 10 entiers, en afficher le nombre de valeurs négatives

```
program CombienNegatif;
var nombre : integer; // le nombre lu au clavier
    compteur : integer; // pour compter 10 nombres
    compteurNegatif : integer; // le nombre de nombre négatifs
begin
    // Avec la boucle WHILE
    compteurNegatif := 0;
    compteur := 1;
    while compteur <= 10 do
    begin
        Write('Quel est le nombre ? ');
        Readln(nombre);
        if nombre < 0 then
            compteurNegatif := compteurNegatif + 1;
        compteur += 1;
    end;
    // Affichage
    Writeln('Avec WHILE, le nombre de negatifs est ',compteurNegatif);

    // Avec la boucle REPEAT
    compteurNegatif := 0;
    compteur := 1;
    repeat
        Write('Quel est le nombre ? ');
        Readln(nombre);
        if nombre < 0 then
            compteurNegatif := compteurNegatif + 1;
        compteur += 1;
    until compteur > 10;
    // Affichage
    Writeln('Avec REPEAT, le nombre de negatifs est ',compteurNegatif);

    // Avec la boucle FOR
    compteurNegatif := 0;
    for compteur := 1 to 10 do
    begin
        Write('Quel est le nombre ? ');
        Readln(nombre);
        if nombre < 0 then
            compteurNegatif := compteurNegatif + 1;
    end;
    // Affichage
    Writeln('Avec FOR, le nombre de negatifs est ',compteurNegatif);

    Readln;
end.
```

3. Calculer le produit de réels lus au clavier. On arrête dès qu'on a lu le réel 0.

```
program ProduitNombres;
var nombre,produit : real; // Le nombre lu au clavier et le produit
begin
    // Avec la boucle WHILE
    produit := 1; //Le produit est 1 car c'est le neutre pour la multiplication
    // Lecture du premier nombre
    Write('Quel est le premier nombre ? ');
    Readln(nombre);
    // Calcul du produit et lecture des autres nombres
    while nombre <> 0 do
    begin
        produit := produit * nombre;
        Write('Quel est le nombre suivant ? ');
        Readln(nombre);
    end;
    // Affichage du produit
    Writeln('Avec WHILE, le produit est ',produit:0:3);

    // Avec la boucle REPEAT
    produit := 1; // Le produit est 1 car c'est le neutre pour la multiplication
    repeat
        Write('Quel est le nombre a multiplier ? ');
        Readln(nombre);
        // On multiplie le nombre au produit si le nombre est différent de 0
        if nombre <> 0 then
        begin
            produit := produit * nombre;
        end;
    until nombre = 0;
    // Affichage du produit
    Writeln('Avec REPEAT, le produit est ',produit:0:3);

    // Il est impossible de faire cet exercice avec la boucle FOR car on ne connaît
    // pas le nombre d'itérations de la boucle

    Readln;
end.
```

4. Calculer la somme des n premiers nombres pairs.

```
program SommeNPremiersPairs;
var n,              // Le nombre d'entiers
    nombre,         // Le nombre ajouté
    somme  : integer; // La somme
begin
    // Lecture du nombre d'entiers
    Write('Combien d''entiers voulez-vous dans la somme ? ');
    Readln(n);

    // Calcul de la somme avec WHILE DO
    somme  := 0;   // Au départ, la somme vaut 0
    nombre := 1;   // On commence par le premier terme
    while nombre <= n do   // Le dernier terme est le ennième
    begin
        somme  := somme + nombre*2;   // On ajoute le nombre^e nombre pair à la somme
        nombre := nombre + 1;          // On passe au terme suivant
    end;
    //Affichage de la somme
    Writeln('Avec la boucle WHILE, la somme est ',somme);

    //Calcul de la somme avec REPEAT UNTIL
    somme  := 0;   // Au départ, la somme vaut 0
    nombre := 1;   // On commence par le premier terme
    repeat
        somme  := somme + nombre*2;   //On ajoute le nombre^e nombre pair à la somme
        nombre := nombre + 1;   //On passe au terme suivant
    until nombre > n;   //Le dernier terme est le n^e
    //Affichage de la somme
    Writeln('Avec la boucle REPEAT, la somme est ',somme);

    //Calcul de la somme avec FOR TO
    somme := 0;   //Au départ, la somme vaut 0
    for nombre := 1 to n do   // Il y a n termes dans le somme
    begin
        somme := somme + nombre*2;   // On ajoute le nombre^e nombre pair à la somme
    end;   // nombre est incrémenté automatiquement
    //Affichage de la somme
    Writeln('Avec la boucle FOR, la somme est ',somme);

    Readln;
end.
```

5. Ce programme calcule la somme des n premiers nombres impairs.

```
program SommeNPremiersImpairs;
var n,        // Le nombre d'entiers
    nombre,   // Le nombre ajouté
    somme   : integer;// La somme
begin
    // Lecture du nombre d'entiers
    Write('Combien d''entiers voulez-vous dans la somme ? ');
    Readln(n);

    // Calcul de la somme avec WHILE DO
    somme  := 0;   //Au départ, la somme vaut 0
    nombre := 1;   //On commence par le premier terme
    while nombre <= n do   // Le dernier terme est le ennième
    begin
        somme  := somme + nombre*2 - 1;   // On ajoute le nombre^e nombre impair à la somme
        nombre := nombre + 1;   // On passe au terme suivant
    end;
    // Affichage de la somme
    Writeln('Avec la boucle WHILE, la somme est ',somme);

    // Calcul de la somme avec REPEAT UNTIL
    somme  := 0;   // Au départ, la somme vaut 0
    nombre := 1;   // On commence par le premier terme
    repeat
        somme  := somme  + nombre*2 - 1;   // On ajoute le nombre^e nombre impair à la somme
        nombre := nombre + 1;   //On passe au terme suivant
    until nombre>n;   //Le dernier terme est le ennième
    // Affichage de la somme
    Writeln('Avec la boucle REPEAT, la somme est ',somme);

    // Calcul de la somme avec FOR TO
    somme:=0;   // Au départ, la somme vaut 0
    for nombre := 1 to n do   // Il y a n termes dans la somme
    begin
        somme := somme + nombre*2 - 1;   // On ajoute le nombre^e nombre impair à la somme
    end;   //nombre est incrémenté automatiquement
    //Affichage de la somme
    Writeln('Avec la boucle FOR, la somme est ',somme);

    Readln;
end.
```

6. Afficher tous les diviseurs d'un nombre (entier).

```
program DiviseursNombre;
var nombre, diviseur : integer;   // Le nombre et un diviseur
begin
    // Lecture du nombre
    Write('Quel est le nombre ? ');
    Readln(nombre);

    // Avec la boucle WHILE
    Writeln;
    Writeln('Avec la boucle WHILE');
    //Affichage des diviseurs
    diviseur := 1; // Le premier diviseur est 1
    // On essaie tous les diviseurs entre 1 et le nombre
    while diviseur <= nombre do
    begin
        // On vérifie si le diviseur testé est bien un diviseur du nombre
        if nombre mod diviseur = 0 then // Le reste de la division doit être 0
        begin
            Writeln(diviseur,' est un diviseur de ',nombre);
        end;
        diviseur := diviseur + 1; // On passe au diviseur suivant
    end;

    //Avec la boucle REPEAT
    Writeln;
    Writeln('Avec la boucle REPEAT');
    // Affichage des diviseurs
    diviseur := 1; // Le premier diviseur est 1
    //On essaie tous les diviseurs entre 1 et le nombre
    repeat
        // On vérifie si le diviseur testé est bien un diviseur du nombre
        if nombre mod diviseur = 0 then // Le reste de la division doit être 0
        begin
            Writeln(diviseur,' est un diviseur de ',nombre);
        end;
        diviseur := diviseur + 1; //On passe au diviseur suivant
    until diviseur > nombre;

    //Avec la boucle FOR
    Writeln;
    Writeln('Avec la boucle FOR');
    // Affichage des diviseurs
    for diviseur := 1 to nombre do
    begin
        // On vérifie si le diviseur testé est bien un diviseur du nombre
        if nombre mod diviseur = 0 then // Le reste de la division doit être 0
        begin
            Writeln(diviseur,' est un diviseur de ',nombre);
        end;
    end;

    Readln;
end.
```

7. Compter le nombre de diviseurs d'un nombre.

```
program DiviseursNombre;
var nombre, diviseur : integer;   // Le nombre et un diviseur
    compteur : integer;   // Le nombre de diviseurs
begin
    //Lecture du nombre
    Write('Quel est le nombre ? ');
    Readln(nombre);

    //Avec la boucle WHILE
    Writeln;
    Writeln('Avec la boucle WHILE');
    // Comptage des diviseurs
    compteur := 0;   // On n'a pas encore trouvé de diviseurs
    diviseur := 1;   // Le premier diviseur est 1
    // On essaie tous les diviseurs entre 1 et le nombre
    while diviseur <= nombre do
    begin
        // On vérifie si le diviseur testé est bien un diviseur du nombre
        if nombre mod diviseur = 0 then // Le reste de la division doit être 0
        begin
            compteur:=compteur+1;
        end;
        diviseur := diviseur + 1; // On passe au diviseur suivant
    end;
    //Affichage du nombre de diviseurs
    Writeln(nombre,' a ',compteur ,' diviseurs');

    // Avec la boucle REPEAT
    Writeln;
    Writeln('Avec la boucle REPEAT');
    //Comptage des diviseurs
    compteur := 0;   // On n'a pas encore trouvé de diviseurs
    diviseur := 1;   // Le premier diviseur est 1
    //On essaie tous les diviseurs entre 1 et le nombre
    repeat
        // On vérifie si le diviseur testé est bien un diviseur du nombre
        if nombre mod diviseur = 0 then // Le reste de la division doit être 0
        begin
            compteur:=compteur+1;
        end;
        diviseur := diviseur + 1; // On passe au diviseur suivant
    until diviseur>nombre;
    //Affichage du nombre de diviseurs
    Writeln(nombre,' a ',compteur ,' diviseurs');
```

```
    // Avec la boucle FOR
    Writeln;
    Writeln('Avec la boucle FOR');
    // Comptage des diviseurs
    compteur := 0;  // On n'a pas encore trouvé de diviseurs
    for diviseur := 1 to nombre do
    begin
        // On vérifie si le diviseur testé est bien un diviseur du nombre
        if nombre mod diviseur = 0 then // Le reste de la division doit être 0
        begin
            compteur := compteur + 1;
        end;
    end;
    // Affichage du nombre de diviseurs
    Writeln(nombre,' a ',compteur ,' diviseurs');

    Readln;
end.
```

8. Calculer le plus grand et le plus petit parmi n nombres réels.

```
program PlusGrandPlusPetit;
var n,    // Le nombre de nombres
    compteur : integer; //Le nombre de nombres déjà entrés
    nombre, // Le nombre entré
    min,max : real;  // Le plus petit et le plus grand
begin
    //Lecture du nombre de nombres
    Write('Combien de nombres allez-vous entrer ? ');
    Readln(n);
    Writeln;

    //Avec la boucle WHILE
    Write('Quel est le premier nombre ? ');
    Readln(nombre);
    min := nombre; // Le plus petit est provisoirement le seul entré
    max := nombre; // Le plus grand est provisoirement le seul entré
    compteur := 1; // On a entré un seul nombre
    // Le programme lit les nombres suivants et cherche le minimum et le maximum
    while compteur < n do
    begin
        // Lecture du nombre suivant
        Write('Quel est le nombre suivant ? ');
        Readln(nombre);
        // On vérifie si c'est le nombre entré qui est le plus petit
        if nombre < min then
        begin
            min := nombre;
        end;
        // On vérifie si c'est le nombre entré qui est le plus grand
        if nombre > max then
        begin
            max := nombre;
        end;
        compteur := compteur + 1;  //On a encodé un nombre de plus
    end;
```

```
// Affichage du minimum et du maximum
Writeln;
Writeln('Avec la boucle WHILE');
Writeln('Le plus petit est ',min:0:3);
Writeln('Le plus grand est ',max:0:3);
Writeln;

// Avec la boucle REPEAT
min := 1.0E+300;  // Le plus petit est provisoirement un nombre très grand
max := -1.0E+300; // Le plus grand est provisoirement un nombre très petit
compteur := 0; //On n'a encore rien entré

// On encode les nombres suivants et on cherche le minimum et le maximum
repeat
    // Lecture du nombre suivant
    Write('Quel est le nombre ? ');
    Readln(nombre);
    // On vérifie si c'est le nombre entré qui est le plus petit
    if nombre < min then
    begin
        min:=nombre;
    end;
    // On vérifie si c'est le nombre entré qui est le plus grand
    if nombre > max then
    begin
        max:=nombre;
    end;
    compteur := compteur + 1;  // On a encodé un nombre de plus
until compteur = n;
//Affichage du minimum et du maximum
Writeln;
Writeln('Avec la boucle REPEAT');
Writeln('Le plus petit est ',min:0:3);
Writeln('Le plus grand est ',max:0:3);
Writeln;

//Avec la boucle FOR
min := 1.0E+300;  // Le plus petit est provisoirement un nombre très grand
max := -1.0E+300; // Le plus grand est provisoirement un nombre très petit
//On encode les nombres suivants et on cherche le minimum et le maximum
for compteur := 1 to n do
begin
    // Lecture du nombre suivant
    Write('Quel est le nombre ? ');
    Readln(nombre);
    // On vérifie si c'est le nombre entré qui est le plus petit
    if nombre < min then
    begin
        min := nombre;
    end;
    // On vérifie si c'est le nombre entré qui est le plus grand
    if nombre > max then
    begin
        max := nombre;
    end;
end;
```

```
    //Affichage du minimum et du maximum
    Writeln;
    Writeln('Avec la boucle FOR');
    Writeln('Le plus petit est ',min:0:3);
    Writeln('Le plus grand est ',max:0:3);
    Writeln;

    Readln;
end.
```

9. Calculer le plus grand et le plus petit des nombres entrés. On arrête si le nombre entré est 0.

```
program PlusGrandPlusPetit;
var nombre, // Le nombre entré
    min, max : real;   // Le plus petit et le plus grand
begin
    //Avec la boucle WHILE
    Write('Quel est le premier nombre ? ');
    Readln(nombre);

    min := nombre; // Le plus petit est provisoirement le seul entré
    max := nombre; // Le plus grand est provisoirement le seul entré

    // On encode les nombres suivants et on cherche le minimum et le maximum
    while nombre <> 0 do
    begin
        // Lecture du nombre suivant
        Write('Quel est le nombre suivant ? ');
        Readln(nombre);
        // On vérifie si c'est le nombre entré (non nul) qui est le plus petit
        if nombre <> 0 then
        begin
            if nombre < min then
            begin
                min := nombre;
            end;
            //On vérifie si c'est le nombre entré (non nul) qui est le plus grand
            if nombre > max then
            begin
                max := nombre;
            end;
        end;
    end;
    //Affichage du minimum et du maximum
    Writeln;
    Writeln('Avec la boucle WHILE');
    Writeln('Le plus petit est ',min:0:3);
    Writeln('Le plus grand est ',max:0:3);
    Writeln;
```

```
    // Avec la boucle REPEAT
    min := 1.0E+300;  // Le plus petit est provisoirement un nombre très grand
    max := -1.0E+300; // Le plus grand est provisoirement un nombre très petit
    // On encode les nombres suivants et on cherche le minimum et le maximum
    repeat
        // Lecture du nombre suivant
        Write('Quel est le nombre ? ');
        Readln(nombre);
        // On vérifie si c'est le nombre entré (non nul) qui est le plus petit
        if nombre <> 0 then
        begin
            if nombre < min then
            begin
                min := nombre;
            end;
            //On vérifie si c'est le nombre entré (non nul) qui est le plus grand
            if nombre > max then
            begin
                max := nombre;
            end;
        end;
    until nombre = 0;
    //Affichage du minimum et du maximum
    Writeln;
    Writeln('Avec la boucle REPEAT');
    Writeln('Le plus petit est ',min:0:3);
    Writeln('Le plus grand est ',max:0:3);
    Writeln;

    //Il est impossible d'utiliser la boucle FOR car on ne connaît pas le nombre d'itérations

    Readln;
end.
```

10. Afficher le nombre de diviseurs de tous les entiers de 10 à 100.

```
program DiviseursNombre;
var nombre, diviseur : integer;   // Le nombre et un diviseur
    compteur : integer;           // Le nombre de divisuers
begin
    // Avec la boucle WHILE
    Writeln('Avec la boucle WHILE');
    // Comptage des diviseurs
    nombre := 10; // On commence avec le nombre 10
    while nombre <= 100 do // Le dernier nombre est 100
    begin
        compteur := 0;  // On n'a pas encore trouvé de diviseurs
        diviseur := 1;  // Le premier diviseur est 1
```

```
        // On essaie tous les diviseurs entre 1 et le nombre
        while diviseur <= nombre do
        begin
            // On vérifie si le diviseur testé est bien un diviseur du nombre
            if nombre mod diviseur = 0 then // Le reste de la division doit être 0
            begin
                compteur := compteur+1;
            end;
            diviseur := diviseur+1; // On passe au diviseur suivant
        end;
        // Affichage du nombre de diviseurs
        Writeln(nombre:3,' a ',compteur:2 ,' diviseurs');
        nombre:=nombre+1; // On passe au nombre suivant
    end;

    // Avec la boucle REPEAT
    Writeln('Avec la boucle REPEAT');
    nombre := 10; // On commence avec le nombre 10
    repeat
        // Comptage des diviseurs
        compteur := 0;  // On n'a pas encore trouvé de diviseurs
        diviseur := 1;  // Le premier diviseur est 1
        // On essaie tous les diviseurs entre 1 et le nombre
        repeat
            // On vérifie si le diviseur testé est bien un diviseur du nombre
            if nombre mod diviseur = 0 then // Le reste de la division doit être 0
            begin
                compteur := compteur + 1;
            end;
            diviseur := diviseur + 1; // On passe au diviseur suivant
        until diviseur > nombre;
        //Affichage du nombre de diviseurs
        Writeln(nombre:3,' a ',compteur:2 ,' diviseurs');
        nombre := nombre+1; // On passe au nombre suivant
    until nombre > 100;

    // Avec la boucle FOR
    Writeln('Avec la boucle FOR');
    for nombre := 10 to 100 do
    begin
        // Comptage des diviseurs
        compteur := 0;  // On n'a pas encore trouvé de diviseurs
        for diviseur := 1 to nombre do
        begin
            // On vérifie si le diviseur testé est bien un diviseur du nombre
            if nombre mod diviseur = 0 then // Le reste de la division doit être 0
            begin
                compteur := compteur + 1;
            end;
        end;
        //Affichage du nombre de diviseurs
        Writeln(nombre:3,' a ',compteur:2 ,' diviseurs');
    end;

    Readln;
end.
```

11. Afficher le nombre de diviseurs de tous les entiers de 10 à 100. Il affiche aussi celui qui a le plus de diviseurs.

> **Remarque :** Quand on connaît le nombre d'étapes à exécuter, la boucle FOR est souvent la plus simple à utiliser. Ainsi, les exercices qui suivent utiliseront cette boucle quand cela sera possible. Transformer la boucle FOR en utilisant une boucle WHILE ou REPEAT est laissé à titre d'exercices.

```
program DiviseursNombre;
var nombre, diviseur : integer;  // Le nombre et un diviseur
    compteur : integer;  // Le nombre de diviseurs
    nombreMax, max : integer; // Le nombre qui a le plus de diviseurs et son nombre de diviseurs
begin
    max := 0;
    for nombre := 10 to 100 do
    begin
        // Comptage des diviseurs
        compteur := 0;  // On n'a pas encore trouvé de diviseurs
        for diviseur := 1 to nombre do
        begin
            // On vérifie si le diviseur testé est bien un diviseur du nombre
            if nombre mod diviseur = 0 then // Le reste de la division doit être 0
            begin
                compteur := compteur + 1;
            end;
        end;

        // Affichage du nombre de diviseurs
        Writeln(nombre:3,' a ',compteur:2 ,' diviseurs');
        // On vérifie si le nombre contient le plus de diviseurs
        if compteur > max then
        begin
            max := compteur;
            nombreMax := nombre;
        end;
    end;

    // Affichage du niombre qui a le plus de diviseurs
    Writeln ('C''est ', nombreMax,
             ' qui a le plus de diviseurs, il en a ',max);

    Readln;
end.
```

12. Déterminer si un nombre est premier.

```
program NombrePremier;
var nombre, diviseur : integer;   // Le nombre et un diviseur
    compteur : integer;   // Le nombre de diviseurs
begin
    // Lecture du nombre
    Write('Quel est le nombre ? ');
    Readln(nombre);

    // Comptage des diviseurs
    compteur := 0;   // On n'a pas encore trouvé de diviseurs
    for diviseur := 1 to nombre do
    begin
        // On vérifie si le diviseur testé est bien un diviseur du nombre
        if nombre mod diviseur = 0 then // Le reste de la division doit être 0
        begin
            compteur := compteur+1;
        end;
    end;

    //Affichage du résultat
    if compteur = 2 then
    begin
        Writeln(nombre,' est un nombre premier');
    end
    else
    begin
        Writeln(nombre,' n''est pas un nombre premier');
    end;

    Readln;
end.
```

$\Longleftrightarrow$

```
program NombrePremier;
var nombre, diviseur : integer;   // Le nombre et un diviseur
    compteur : integer;   //Le nombre de diviseurs
begin
    // Lecture du nombre
    Write('Quel est le nombre ? ');
    Readln(nombre);

    // Comptage des diviseurs
    compteur := 0;   // On n'a pas encore trouvé de diviseurs
    for diviseur := 1 to nombre do
        if nombre mod diviseur = 0 then // Le reste de la division doit être 0
            compteur := compteur+1;

    //Affichage du résultat
    if compteur = 2 then
        Writeln(nombre,' est un nombre premier')
    else
        Writeln(nombre,' n''est pas un nombre premier');
    Readln;
end.
```

13. Déterminer si un nombre est parfait (il est égal à la somme de ses diviseurs excepté lui-même).

```
program NombreParfait;
var nombre, diviseur : integer;   // Le nombre et un diviseur
    somme : integer;   // La somme des diviseurs
begin
    // Lecture du nombre
    Write('Quel est le nombre ? ');
    Readln(nombre);

    //Comptage des diviseurs
    somme := 0;   // On n'a pas encore trouvé de diviseurs
    for diviseur := 1 to nombre-1 do
        //On vérifie si le diviseur testé est bien un diviseur du nombre
        if nombre mod diviseur = 0 then // Le reste de la division doit être 0
            somme := somme + diviseur; // On ajoute le diviseur à la somme

    // Affichage du résultat
    if somme = nombre then
        Writeln(nombre,' est un nombre parfait')
    else
        Writeln(nombre,' n''est pas un nombre parfait');

    Readln;
end.
```

14. Calculer a^n (n entier positif)

```
program aExposantN;
var a : real; // La base de la puissance
    n : integer; // L'exposant de la puissance
    puissance : real; // La puissance a exposant n
    compteur : integer; / /Le nombre de fois que la base àéé t émultiplie
begin
    // Lecture de la base et de l'exposant
    Write('Quelle est la base ? ');
    Readln(a);
    Write('Quel est l''exposant ? ');
    Readln(n);

    // Calcul de la puissance
    puissance := 1; //Le neutre pour le produit
    //On multiplie n fois puissance par a
    for compteur := 1 to n do
        puissance := puissance * a;

    //Affichage du résultat
    Writeln;
    Writeln('La puissance est ',puissance:0:3);

    Readln;
end.
```

15. Calculer a^n (n entier quelconque).

Rappels :

$$a^n = \frac{1}{a^{-n}} \quad \text{et} \quad a^0 = 1 \quad \text{pour} \quad a \neq 0$$

```
program aExposantN;
var a : real; // La base de la puissance
    n : integer; // L'exposant de la puissance
    puissance : real; // La puissance a exposant n
    compteur : integer; // Le nombre de fois que la base à été multipliée
begin
    // Lecture de la base et de l'exposant
    Write('Quelle est la base ? ');
    Readln(a);
    Write('Quel est l''exposant ? ');
    Readln(n);

    if n = 0 then
        if a = 0 then
            Writeln ('Indetermination')
        else
            Writeln('La puissance est 1')
    else
    begin
        // Calcul de la puissance
        puissance := 1; // Le neutre pour le produit
        // On multiplie abs(n) fois puissance par a (abs est la valeur absolue.)
        for compteur := 1 to abs(n) do
            puissance := puissance * a;

        // Si l'exposant est négatif, on prends l'inverse
        if n < 0 then
            puissance : = 1/puissance;

        // Affichage du résultat
        Writeln;
        Writeln('La puissance est ',puissance:0:3);
    end;

    Readln;
end.
```

16. Calculer la probabilité d'obtenir un 6 avec un dé.

```
program Probabilite6;
var lance,       // Résultat d'un lancé de dé
    compteur,    // Compteur de lancés de dé
    nombre6: integer; // Nombre de 6 obtenus
    probabilite : real;  // Inverse de la probabilité d'obtenir 6
begin
    randomize; // Initiation du générateur aléatoire

    // On simule 1 000 000 de lancés
    nombre6 := 0; // On n'a pas encore obtenu de 6
    for compteur := 1 to 1000000 do
    begin
        lance := 1 + random(6); // On lance un dé
        if lance = 6 then
            nombre6 := nombre6 + 1; // On compte le 6 obtenu
    end;

    // Calcul de l'inverse de la probabilité
    probabilite := 1000000 / nombre6;

    // Affichage de la probabilité
    Writeln('On a 1 chance sur ',probabilite:0:2,' d''obtenir 6');
    Writeln;

    Readln;
end.
```

17. Afficher tous les nombres d'Armstrong. (Un nombre est d'Armstrong si la somme des cubes de ses trois chiffres est égale au nombre).

```
program Armstrong;
var nombre : integer;    // Le nombre de trois chiffres
    u,d,c : integer;     // Le chiffre des unités, des dizaines et des centaines
begin
    for nombre := 100 to 999 do // 100 est le premier et 999 est le dernier nombre de 3 chiffres
    begin
        // Décomposition du nombre en trois chiffres
        u := nombre mod 10;
        d := nombre div 10 mod 10;
        c := nombre div 100;

        // On regarde si le nombre est d'Armstrong
        if u*u*u + d*d*d + c*c*c = nombre then
            Writeln(nombre,' est un nombre d''Armstrong !');
    end;

    Readln;
end.
```

18. Ce programme affiche tous les nombres premiers entre 2 et 1000.

```
program NombrePremier;
var nombre, diviseur : integer;   // Le nombre et un diviseur
    compteur : integer;   // Le nombre de diviseurs
begin
    for nombre := 2 to 1000 do
    begin
        // Comptage des diviseurs
        compteur:=0;   //On n'a pas encore trouvé de diviseurs
        for diviseur := 1 to nombre do
            // On vérifie si le diviseur testé est bien un diviseur du nombre
            if nombre mod diviseur = 0 then // Le reste de la division doit être 0
                compteur := compteur + 1;

        // Affichage du résultat
        if compteur = 2 then
            Writeln(nombre,' est un nombre premier');
    end;

    Readln;
end.
```

19. Afficher tous les nombres parfaits de 1 à 10000.

```
program NombreParfait;
var nombre, diviseur : integer;   // Le nombre et un diviseur
    somme : integer;   // La somme des diviseurs
begin
    for nombre := 1 to 10000 do
    begin
        // Comptage des diviseurs
        somme := 0;   // On n'a pas encore trouvé de diviseurs
        for diviseur := 1 to nombre-1 do
            // On vérifie si le diviseur testé est bien un diviseur du nombre
            if nombre mod diviseur = 0 then // Le reste de la division doit être 0
                somme := somme + diviseur; // On ajoute le diviseur à la somme

        // Affichage du résultat
        if somme = nombre then
            Writeln(nombre,' est un nombre parfait');
    end;

    Readln;
end.
```

20. Afficher tous les couples d'entiers entre -100 et 100 vérifiant la relation $9x-4y=35$

```
program couples;
var x,y:integer; //Les valeurs de x et y
begin
   Writeln('Voici les couples qui verifient la relation 9x-4y=35 :');
   for x := -100 to 100 do
      for y := -100 to 100 do
         if 9*x - 4*y = 35 then
            Writeln('(',x:3,';',y:3,')');

   Readln;
end.
```

21. Afficher tous les triplets de nombres entiers entre 1 et 100 qui vérifient la relation de Pythagore (ils vérifient la relation $x^2 + y^2 = z^2$).

```
program TripletsPythagore;
var x,y,z : integer;  // Le triplet
begin
   Writeln('Voici les triplets de Pythagore :');

   for x := 1 to 100 do
      for y := 1 to 100 do
         for z := 1 to 100 do
            if x*x + y*y = z*z then
               Writeln('(',x:3,';',y:3,';',z:3,')');

   Readln;
end.
```

22. Calculer le capital obtenu en plaçant un capital initial à intérêts simples et à intérêts composés.

```
program interets;
var Cf,Cd,t : real; // Le capital final, le capital de départ et le taux
   n : integer; // Le nombre d'années
   i : integer;
begin
   Write('Capital de depart ? ');
   Readln(Cd);
   Write('Combien d''annees ? ');
   Readln(n);
   Write('Taux interets (en %) ? ');
   Readln(t);

   t := t/100;
```

```
    // Intérêts simples
    Cf := Cd*(1+n*t);
    Writeln('Interets simples = ', Cf:0:2);

    // Intérêts composes
    // Calculer (1+t)^n
    Cf := 1;
    for i:=1 to n do
        Cf := Cf * (1+t);
    // Multiplier par Cd
    Cf := Cd * Cf;

    Writeln('Interets composes = ', Cf:0:2);

    Readln;
end.
```

23. Programmer le jeu de la fourchette.

```
program NombreAleatoire;
var nombreOrdinateur, nombreUtilisateur, nombreEssais, essaiMax : integer;
begin
    randomize;

    essaiMax := 0;
    nombreOrdinateur := random(1000); // 0..999
    essaiMax := 10;

    nombreEssais := 1;
    Write('Quelle est votre proposition ? ');
    Readln(nombreUtilisateur);

    while (nombreEssais <= essaiMax) and
          (nombreUtilisateur <> nombreOrdinateur) do
    begin
        if nombreUtilisateur < nombreOrdinateur then
            Writeln('Trop petit!')
        else
            Writeln('Trop grand!');
        Write('Quelle est votre proposition ? ');
        Readln(nombreUtilisateur);
        nombreEssais := nombreEssais + 1;
    end;

    if nombreEssais <= essaiMax then
        Writeln('Bravo, vous avez trouve en ',nombreEssais,' essai !')
    else
        Writeln('Le nombre a trouver etait ', nombreOrdinateur);

    Readln;
end.
```

24. Calculer le nombre de 'e' d'une chaîne. (Il faut tenir compte des majuscules et des minuscules).

```
program NombreE;
const VOYELLES = 'aeiouyAEIOUY';
var ch : String;
  i : Integer;
  cpt : Integer;
begin
  Write('Quelle est la chaine ?');
  ReadLn(ch);

  cpt := 0;
  for i:=1 to Length(ch) do
      if Pos(ch[i],VOYELLES) > 0 then // le ieme caractère est-il une voyelle ?
         cpt := cpt + 1;

  WriteLn(ch, ' contient ', cpt, ' voyelles');

  ReadLn;
end.
```

25. Calculer le nombre de voyelles, de consonnes et des autres caractères d'une chaîne.

```
program NombreChar;
const VOYELLES = 'AEIOUY';
  CONSONNES = 'ZRTPQSDFGHJKLMWXCVBN';
var ch : String;
  i : Integer;
  cptVoy, cptCons, cptAutres : Integer;
begin
  Write('Quelle est la chaine ?');
  ReadLn(ch);

  ch := Upcase(ch);// Mettre la chaine en majuscules

  cptVoy:=0; cptCons:=0; CptAutres:=0;
  for i:=1 to Length(ch) do
      if Pos(ch[i],VOYELLES) > 0 then // le ieme caractère est-il une voyelle ?
         cptVoy := cptVoy + 1
      else
        if Pos(ch[i],CONSONNES) > 0 then // le ieme caractère est-il une consonne ?
           cptCons := cptCons + 1
        else // C'est que c'est autre chose
           cptAutres := cptAutres + 1;

  WriteLn(ch, ' contient ', cptVoy, ' voyelles, ', cptCons, ' consonnes et ',
              cptAutres, ' autres caracteres');

  ReadLn;
end.
```

26. Remplacer toutes les minuscules d'une chaîne par des étoiles (*).

```
program RemplacerMinEtoile;
const MINUSCULES = 'azertyuiopqsdfghjklmwxcvbn';
var chDepart, chFin : String;
  i : Integer;
begin
  Write('Quelle est la chaine ?');
  ReadLn(chDepart);

  chFin := '';       // La seconde chaine est vide au départ
  for i:=1 to Length(chDepart) do
     // le ieme caractère est-il une minuscule ?
      if Pos(chDepart[i],MINUSCULES) > 0 then
         chFin := chFin + '*'
      else
         chFin := chFin + chDepart[i];

  WriteLn(chDepart, ' est devenue ', chFin);

  ReadLn;
end.
```

27. Renverser une chaîne.

```
program RenverseChaine;
var chDepart, chFin : String;
  i : Integer;
begin
  Write('Quelle est la chaine ?');
  ReadLn(chDepart);

  chFin := '';       // La seconde chaine est vide au départ
  for i:=Length(chDepart) downto 1 do
      chFin := chFin + chDepart[i];

  WriteLn(chDepart, ' renversee est ', chFin);

  ReadLn;
end.
```

28. Compter le nombre de mots d'une chaîne. On suppose qu'il n'y a pas de lettres accentuées.

```
program CompterMots;
const LETTRES = 'azertyuiopqsdfghjklmwxcvbnAZERTYUIOPQSDFGHJKLMWXCVBN';
var   nMots: integer;
   chaine: string;
   i: Integer;
   nouveauMot : boolean;
begin
    Write('Quelle est la chaine ? ');
    ReadLn(chaine);

   // Comptage
   nMots := 0;
   nouveauMot := False;
   for i := 1 to Length(chaine) do
   begin
      // Pour chaque caractère, on regarde si c'est une lettre
      // et qu'on n'est pas entrain de traiter un nouveau mot
       if (Pos(chaine[i], LETTRES) > 0) and (nouveauMot = False) then
       begin
          inc(nMots); // Si on n'est pas déjà entrain de traiter un mot,
                      // on augmente le compteur de mots
          nouveauMot := True;// On est entrain de traiter un nouveau mot
       end
       else
          if Pos(chaine[i], LETTRES) = 0 then // Le caractère n'est plus une lettre
             nouveauMot := False;
   end;

   Write('Il y a ', nMots, ' mots');

   ReadLn;
end.
```

Une autre solution est de partir de l'idée qu'on a un nouveau mot si une lettre est suivie d'un caractère qui n'est pas une lettre (on traite $chaine[i]$ et $chaine[i+1]$ en même temps).

CHAPITRE 2

TABLEAUX | SOUS-PROGRAMMES | FICHIERS

2.1 Les tableaux

1. Multiplier tous les éléments d'un tableau par 4.

```
program TableauEx1;
// Le tableau est lu au clavier.
var T : array [1..6] of integer;
    i : integer;
begin
    for i:=1 to 6 do
    begin
        write (' Entrez un nombre entier: ');
        readln (T[i]);
    end;

    // Afficher le tableau de départ
    write (' T1: ');
    for i:=1 to 6 do
        write (T[i], '  ');
    writeln;

    // Multiplication par 4
    for i:=1 to 6 do
       T[i] := T[i] * 4;

    // Afficher le tableau modifié
    write (' Tx4: ');
    for i := 1 to 6 do
        write (T[i],' ');

    readln;
end.
```

2. Calculer la somme de deux tableaux à une dimension (utiliser trois tableaux).

```
program tableaux2;
var T1, T2, T3 : array[1..6] of integer;
    i : integer;
begin
    // Lecture du premier tableau au clavier
    for i := 1 to 6 do
    begin
        write ('Entrez un nombre pour le premier tableau: ');
        readln (T1[i]);
    end;

    // Lecture du second tableau au clavier
    for i := 1 to 6 do
    begin
        write ('Entrez un nombre pour le deuxieme tableau: ');
        readln (T2[i]);
    end;

    // Somme
    for i := 1 to 6 do
    begin
        T3[i] := T1[i] + T2[i];
    end;

    // Affichages des trois tableaux
    write ('T1= ');
    for i := 1 to 6 do
        write (T1[i],' ' );
    writeln;

    write ('T2= ');
    for i := 1 to 6 do
        write (T2[i],' ' );
    writeln;

    write ('T3= ');
    for i := 1 to 6 do
        write (T3[i],' ' );

    readln;
end.
```

3. Afficher un tableau de N éléments sur M colonnes.

```
program ExTableauMColonnes;
const nMax = 1000;
var tab: array[1..NMax] of integer;
   N, M, i: integer;
begin
   randomize;
   repeat
      write ('Nombre d''elements du tableau (entre 1 et ',nMax,') : ');
      readln(N);
   until (N > 0) and ( N <= nMax);

   write (' Nombre de colonnes pour l''affichage du tableau : ');
   readln(M);

   // Génération aléatoire
   for i := 1 to N do
      tab[i] := 1 + random (100);

   // Affichage sur M colonnes
   for i := 1 to N do
      if i mod M = 0 then
         writeln(tab[i] : 5)
      else
         write(tab[i] :5 );

   readln;
end.
```

4. Inverser le premier et le dernier élément d'un tableau.

```
program TableauEx7;
var tab : array[1..100] of integer;
   i, N, echange : integer;
begin
   randomize;
   write (' Combien d''elements dans le tableau? ');
   readln (N);

   // Génération
   for i:=1 to N do
   begin
      tab [i] := 1 + random(100);
      write(tab[i],' ');
   end;
   writeln;

   // Échange
   echange := tab [N];
   tab [N] := tab [1];
   tab [1] := echange ;
```

```
    // Affichage
    for i:=1 to N do
        write (tab[i], ' ');

    readln;
end.
```

5. Permuter circulairement un tableau vers la gauche.

```
program TableauEx8;
const nMax=1000;
var tab: array[1..nMax] of integer;
    i, N, memoire: integer;
begin
    randomize;
    repeat
        write ('Combien d"elements dans le tableau (entre 1 et ',nMax,') : ');
        readln(N);
    until (N > 0) and (N <= nMax);

    for i := 1 to N do
    begin
        tab [i] := -100 + random(201);
        write(tab[i], ' ');
    end;
    writeln;

    // Permutation vers la gauche
    memoire := tab[1];   // Retenir le premier
    for i := 1 to N-1 do // Décaler
    begin
        tab[i] := tab[i+1];
    end;
    tab[n] := memoire; // Mettre l'élément retenu en dernière place

    // Afficher
    for i := 1 to N do
    begin
        write (tab[i], ' ');
    end;

    readln;
end.
```

6. Permuter circulairement un tableau vers la droite.

```
program TableauEx8;
const nMax=1000;
var tab: array[1..NMax] of integer;
    i, N, memoire: integer;
begin
    randomize;
    repeat
        write ('Combien d''elements dans le tableau (entre 1 et ',nMax,') : ');
        readln(N);
    until (N > 0) and (N <= nMax);

    for i := 1 to N do
    begin
        tab[i] := -100 + random(201);
        write(tab[i], ' ');
    end;
    writeln;

    // Permutation vers la droite (on utilise DownTo)
    memoire := tab[N];   // Retenir le dernier
    for i := N downto 2 do // Décaler
    begin
        tab[i] := tab[i-1];
    end;
    tab[1] := memoire; // Mettre l'élément retenu en première place

    // Afficher
    for i := 1 to N do
    begin
        write (tab[i], ' ');
    end;

    readln;
end.
```

7. Permuter circulairement un tableau d'un nombre quelconque de positions vers la droite ou vers la gauche.

```
program Ex10Tableaux;
var tab : array[1..1000] of integer;
    tab2 :array[1..1000] of integer;
    i, N, p, DG: integer;
begin
    randomize;
    write ('Combien d''elements dans le tableau? ');
    readln (n);
    write ('De combien voulez vous permuter ? ');
    readln (p);
    write (' Dans quel sens ( Gauche = 0 ou Droite = 1 )? ');
    readln (DG);

    // Générer aléatoirement et affichage
    for i:=1 to N do
    begin
        tab [i] := -100+random(201);
        write (tab[i],' ');
    end;
    writeln;

    If DG = 0 then // vers la gauche
    begin
        for i := 1 to N-p do
            tab2[i] := tab[i+p];
        for i := n-p+1 to n do
            tab2[i] := tab[i-(n-p)];
    end
    else  // vers la droite
    begin
        for i := 1 to p do
            tab2[i] := tab[n-p+i];
        for  i := 1 to n-p do
            tab2[i+p] := tab[i];
    end;

    // Affichage
    for i:=1 to N do
        write (tab2[i], ' ');

    readln;
end.
```

8. Créer un tableau contenant les nombres de Fibonacci.

```
program TableauxAvecNombreDeFibonacci;
var tab : array[1..100] of integer;
   i, N : integer;
begin
   randomize;
   write (' Combien d''elements dans le tableau ? ');
   readln (N);

   tab[1] := 1;
   tab[2] := 1;

   For i := 3 to N do
      tab[i] := tab[i-1] + tab[i-2];

   For i:=1 to N do
      write (tab [i],' ');

   readln;
end.
```

9. Calculer combien de fois est un élément dans un tableau.

```
program Ex14Tableaux;
var tab : array[1..100] of integer;
   i, N, compteur : integer;
begin
   randomize;

   write ('Quel est le nombre que vous recherchez ? ');
   readln (n);

   // Génération aléatoire
   for i := 1 to 100 do
   begin
      tab[i] := -10 + random(21);
      if i mod 10 = 0 then
         writeln(tab[i] :5)
      else
         write(tab[i] : 5);
   end;

   compteur := 0;
   for i := 1 to 100 do
      if tab[i] = n then
         compteur += 1;

   writeln;
   write ('Le nombre que vous recherchez est ', compteur, ' fois dans le tableau.');

   readln;
end.
```

10. Calculer le minimum des éléments d'un tableau de réels, donner sa position.

```
program TableauEx8;
const nMax=1000;
var tab: array[1..NMax] of real;
    i, N : integer;
    min : real;
    position : integer;
begin
    randomize;

    repeat
        write ('Combien d"elements dans le tableau (entre 1 et ',nMax,') : ');
        readln(N);
    until (N > 0) and (N <= nMax);

    for i := 1 to N do
    begin
        tab [i] := random * 100;
        write(tab[i]:0:3,' ');
    end;
    writeln;

    // Au départ, le minimum est en première position
    min := tab[1];
    position := 1;
    // Si on rencontre un élément plus petit que min, alors min devient cet élément
    for i := 2 to N do
        if tab[i] < min then
        begin
            min := tab[i];
            position := i;
        end;

    // Afficher le minimum et sa position
    write ('Le minimum est ', min:0:3, ' en position ', position);

    readln;
end.
```

11. Trier un tableau avec la méthode du tri par sélection du minimum.

```
program MinimumEnPremier;
const nMax=1000;
var tab : array[1..NMax] of integer;
   j, i, N, Min : integer;
   position : integer;
begin
   randomize;

   repeat
      write ('Combien d''elements dans le tableau (entre 1 et ',nMax,') : ');
      readln(N);
   until (N > 0) and (N <= nMax);

   for i := 1 to N do
   begin
      tab[i] := -100 + random(100);
      write(tab[i],' ');
   end;
   writeln;

   // Tri
   for j := 1 to n-1 do
   begin
      Min := Tab[j];
      position := j;

      For i := j+1 to N do
      if tab[i] < Min then
      begin
         min := tab [i];
         position := i;
      end;

      // Échanger le minimum et le j^e
      tab[position] := tab[j];
      tab[j] := min ;
   end;

   for i:=1 to n do
      write (tab[i], ' ');

   readln;
end.
```

12. Trier un tableau en utilisant le tri « Bulles ».

```
program TriBulle;
const nMax=1000;
var tab : array[1..NMax] of integer;
    i, N, memoire : integer;
    echange : boolean;
begin
    randomize;

    repeat
        write ('Combien d''elements dans le tableau (entre 1 et ',nMax,') : ');
        readln(N);
    until (N > 0) and (N <= nMax);

    // Génération du tableau et affichage
    for i:=1 to N do
    begin
        tab[i] := 1 + random(100);
        write(tab[i], ' ');
    end;
    writeln;

    // Tri
    repeat
        echange := false;
        For i := 1 to N-1 do
        begin
            if tab[i] > tab[i+1] then
            begin
                memoire  := tab[i];
                tab[i]   := tab[i+1];
                tab[i+1] := memoire;
                echange  := true;
            end;
        end;
    until echange = false;   // Arrêt si pas d'échange

    // Affichage du tableau trié
    for i := 1 to n do
        write (tab[i], ' ');

    readln;
end.
```

13. Trier un tableau en utilisant le tri à peigne.

```
program TriPeigne;
const nMax=1000;
var tab : array[1..NMax] of integer;
   i, N : integer;
   echange : boolean;
begin
   randomize;

   repeat
      write ('Combien d''elements dans le tableau (entre 1 et ',nMax,') : ');
      readln(N);
   until (N > 0) and (N <= nMax);

   // Génération du tableau et affichage
   for i:=1 to N do
   begin
      tab [i] := 1 + random(100);
      write(tab[i],' ');
   end;
   writeln;

   // Tri
   pas := N;
   repeat
      pas := trunc(pas/1.3);
      if pas < 1 then pas := 1;
      fin := True;
      for i:=1 to (N - pas) do
      begin
         if tab[i] > tab[i + pas] then
         begin
            fin := False;
            echange := tab[i];
            tab[i] := tab[i+pas];
            tab[i+pas] := echange;
         end;
      end;
   until (pas = 1 ) and (fin = true);

   // Affichage du tableau trié
   for i := 1 to N do
      write (tab[i], ' ');

   readln;
end.
```

14. Comparer les tris en mesurant le temps nécessaire pour trier un "grand" tableau. Les tris utilisés sont :
    - le tri bulle ;
    - le tri par sélection du minimum ;
    - le tri par sélection du minimum et du maximum ;
    - le tri par insertion ;
    - ~~le tri Shell~~ (le tri à peigne).

    On trie un tableau de réels puis d'entiers.

```
program TriComparaison;
uses dos, crt;

const nMax = 100000; // Nombre d'éléments maximum à trier

var tab1,tab2 : array[1..nMax] of real;
      // On trie tab1 et on retient le tableau de départ dans tab2
    tab3,tab4 : array[1..nMax] of integer;
      // On trie tab3 et on retient le tableau de départ dans tab4
    echange,min,max,insere : real;
    echange1,min1,max1,insere1 : integer;
    temps : array[1..10] of real;
      // Les 5 premiers éléments servent pour le temps de tri d'un tableau de réels;
      // Les 5 derniers éléments servent pour le temps de tri d'un tableau d'entiers;
    i,j,pas,n,placemin,placemax : integer;
    h,m,s,c : word;
      // heure, minute, seconde, centième de seconde
    Total_sec_depart, Total_sec_fin : real;
    fin : boolean;
    f,g : integer;
      // f est le nombre de fois qu'on exécute chaque tri
      // g est l'indice de boucle qui va de 1 à f
    coeff : array[1..5] of real;
      // Rapports entre les temps de tris pour des réels et des entiers

begin
    // Lecture du nombre d'éléments (n)
    repeat
      write('Combien d''elements ? ');
      readln(n);
    until (n > 0) and (n <= nMax);

    // Lecture du nombre de tris à effectuer pour calculer la moyenne des temps
    repeat
      write('Combien de fois voulez-vous trier ? ');
      readln(f);
    until f > 0;

    // Initialisation du tableau des temps
    for i:=1 to 10 do
      temps[i] := 0;
```

```
// On trie f fois un tableau de réels puis d'entiers avec les cinq méthodes
for g := 1 to f do
begin
    // Génération des éléments des tableaux (un de réels et un d'entiers) et les retenir
    Randomize;
    for i := 1 to n do
    begin
        tab1[i] := random*100;
        tab2[i] := tab1[i];
        tab3[i] := round(tab1[i]);
        tab4[i] := tab3[i];
    end;

    // ***************************************************************************
    // Tris de réels
    // ***************************************************************************
    // Calcul de l'heure de départ
    getTime(h,m,s,c);
    Total_sec_depart := h*3600 + m*60 + s + c/100.0;

    // Tri bulle
    j := 1;
    repeat
        fin:=true;
        for i := 1 to n-j do
            if tab1[i] > tab1[i+1] then
            begin
                echange  := tab1[i];
                tab1[i]  := tab1[i+1];
                tab1[i+1]:= echange;
                fin := false;
            end;
        j += 1;
    until fin = true;

    // Calcul de l'heure de fin
    getTime(h,m,s,c);
    Total_sec_fin := h*3600 + m*60 + s + c/100.0;
    temps[1] += Total_sec_fin - Total_sec_depart;

    // On recopie le deuxième tableau dans le premier pour trier le même tableau
    for i := 1 to n do
        tab1[i] := tab2[i];

    // Calcul de l'heure de départ
    getTime(h,m,s,c);
    Total_sec_depart := h*3600 + m*60 + s + c/100.0;
```

```
// Tri par sélection du minimum
for i := 1 to n-1 do
begin
    // Recherche du minimum et de sa place dans le tableau
    min := tab1[i];
    placemin := i;
    for j := i+1 to n do
        if min > tab1[j] then
        begin
            min := tab1[j];
            placemin := j;
        end;
    // Échanger le minimum et le ie élément du tableau
    tab1[placemin] := tab1[i];
    tab1[i] := min;
end;

// Calcul de l'heure de fin
getTime(h,m,s,c);
Total_sec_fin := h*3600 + m*60 + s + c/100.0;
temps[2] += Total_sec_fin - Total_sec_depart;

// On recopie le deuxième tableau dans le premier pour trier le même tableau
for i:=1 to n do
    tab1[i] := tab2[i];

// Calcul de l'heure de départ
getTime(h,m,s,c);
Total_sec_depart := h*3600 + m*60 + s + c/100.0;
```

```
// Tri par sélection du minimum et du maximum.
// Le minimum est placé au début et en même temps, le maximum est placé à la fin
for i := 1 to n div 2 do    // Que jusqu'à la moitié
begin
    // Recherche du minimum et du maximum et de leur place dans le tableau
    min := tab1[i];
    placemin := i;
    max := tab1[i];
    placemax := i;
    for j := i+1 to n-i+1 do
    begin
        if min > tab1[j] then
        begin
            min := tab1[j];
            placemin := j;
        end;
        if max < tab1[j] then
        begin
            max := tab1[j];
            placemax := j;
        end;
    end;
    // Échanger le minimum et le ie élément du tableau
    // Échanger le maximum et le (n-i+1)e élément du tableau
    tab1[placemin] := tab1[i];
    tab1[i] := min;
    if placemax = i then
        placemax := placemin;
    tab1[placemax] := tab1[n-i+1];
    tab1[n-i+1] := max;
end;
// Calcul de l'heure de fin
getTime(h,m,s,c);
Total_sec_fin := h*3600 + m*60 + s + c/100.0;
temps[3] += Total_sec_fin - Total_sec_depart;

// On recopie le deuxième tableau dans le premier pour trier le même tableau
for i:=1 to n do
    tab1[i]:=tab2[i];

// Calcul de l'heure de départ
getTime(h,m,s,c);
Total_sec_depart := h*3600 + m*60 + s + c/100.0;
```

```
// Tri par insertion
for i := 2 to n do
begin
    insere := tab1[i];
    j := i-1;
    while (j >= 1) and (tab1[j] >= insere) do
    begin
        tab1[j+1] := tab1[j];
        j -= 1;
    end;
    tab1[j+1] := insere;
end;

// Calcul de l'heure de fin
getTime(h,m,s,c);
Total_sec_fin := h*3600 + m*60 + s + c/100.0;
temps[4] += Total_sec_fin - Total_sec_depart;

// On recopie le deuxième tableau dans le premier pour trier le même tableau
for i:=1 to n do
    tab1[i] := tab2[i];

// Calcul de l'heure de départ
getTime(h,m,s,c);
Total_sec_depart := h*3600 + m*60 + s + c/100.0;

// Tri Shell
pas:=n;
while pas>1 do
begin
    pas := pas div 2;
    for i := 1 to n-pas do
    begin
        j := i;
        while (j > 0) and (tab1[j] > tab1[j+pas]) do
        begin
            echange := tab1[j];
            tab1[j] := tab1[j+pas];
            tab1[j+pas] := echange;
            j := j-pas;
        end;
    end;
end;

// Calcul de l'heure de fin
getTime(h,m,s,c);
Total_sec_fin := h*3600 + m*60 + s + c/100.0;
temps[5] += Total_sec_fin - Total_sec_depart;
```

```
// ************************************************************************
// Tris d'entiers
// ************************************************************************
// Calcul de l'heure de départ
getTime(h,m,s,c);
Total_sec_depart := h*3600 + m*60 + s + c/100.0;

// Tri bulle
j := 1;
repeat
    fin := true;
    for i := 1 to n-j do
        if tab3[i] > tab3[i+1] then
        begin
            echange1 := tab3[i];
            tab3[i]  := tab3[i+1];
            tab3[i+1]: =echange1;
            fin := false;
        end;
    j += 1;
until fin = true;

// Calcul de l'heure de fin
getTime(h,m,s,c);
Total_sec_fin := h*3600 + m*60 + s + c/100.0;
temps[6] += Total_sec_fin - Total_sec_depart;

// On recopie le deuxième tableau dans le premier pour trier le même tableau
for i:=1 to n do
    tab3[i] := tab4[i];

// Calcul de l'heure de départ
getTime(h,m,s,c);
Total_sec_depart := h*3600 + m*60 + s + c/100.0;
```

```
// Tri par sélection du minimum
for i:=1 to n-1 do
begin
    // Recherche du minimum et de sa place dans le tableau
    min1 := tab3[i];
    placemin := i;
    for j := i+1 to n do
        if min1 > tab3[j] then
        begin
            min1 := tab3[j];
            placemin := j;
        end;
    // Échanger le minimum et le ie élément du tableau considéré
    tab3[placemin] := tab3[i];
    tab3[i] := min1;
end;

// Calcul de l'heure de fin
getTime(h,m,s,c);
Total_sec_fin := h*3600 + m*60 + s + c/100.0;
temps[7] += Total_sec_fin - Total_sec_depart;

// On recopie le deuxième tableau dans le premier pour trier le même tableau
for i:=1 to n do
    tab3[i]:=tab4[i];

// Calcul de l'heure de départ
getTime(h,m,s,c);
Total_sec_depart := h*3600 + m*60 + s + c/100.0;
```

```
// Tri par sélection du minimum et du maximum
for i:=1 to n div 2 do
begin
    // Recherche du minimum et du maximum et de leur place dans le tableau
    min1 := tab3[i];
    placemin := i;
    max1 := tab3[i];
    placemax := i;
    for j := i+1 to n-i+1 do
    begin
        if min1 > tab3[j] then
        begin
            min1 := tab3[j];
            placemin := j;
        end;
        if max1 < tab3[j] then
        begin
            max1 := tab3[j];
            placemax := j;
        end;
    end;
    // Échanger le minimum et le ie élément du tableau
    // Échanger le maximum et le (n-i+1)ie élément du tableau
    tab3[placemin] := tab3[i];
    tab3[i] := min1;
    if placemax=i then
        placemax:=placemin;
    tab3[placemax] := tab3[n-i+1];
    tab3[n-i+1] := max1;
end;

//Calcul de l'heure de fin
getTime(h,m,s,c);
Total_sec_fin := h*3600 + m*60 + s + c/100.0;
temps[8] += Total_sec_fin - Total_sec_depart;

// On recopie le deuxième tableau dans le premier pour trier le même tableau
for i:=1 to n do
    tab3[i] := tab4[i];

// Calcul de l'heure de départ
getTime(h,m,s,c);
Total_sec_depart := h*3600 + m*60 + s + c/100.0;
```

```
    // Tri par insertion
    for i:=2 to n do
    begin
        insere1 := tab3[i];
        j := i-1;
        while (j >= 1) and (tab3[j] >= insere1) do
        begin
            tab3[j+1] := tab3[j];
            j -= 1;
        end;
        tab3[j+1] := insere1;
    end;

    // Calcul de l'heure de fin
    getTime(h,m,s,c);
    Total_sec_fin := h*3600 + m*60 + s + c/100.0;
    temps[9] += Total_sec_fin - Total_sec_depart;

    // On recopie le deuxième tableau dans le premier pour trier le même tableau
    for i := 1 to n do
        tab3[i]:=tab4[i];

    // Calcul de l'heure de départ
    getTime(h,m,s,c);
    Total_sec_depart := h*3600 + m*60 + s + c/100.0;

    // Tri Shell
    pas:=n;
    while pas>1 do
    begin
        pas := pas div 2;
        for i := 1 to n-pas do
        begin
            j := i;
            while (j > 0) and (tab3[j] > tab3[j+pas]) do
            begin
                echange1 := tab3[j];
                tab3[j] := tab3[j+pas];
                tab3[j+pas] := echange1;
                j := j-pas;
            end;
        end;
    end;

    // Calcul de l'heure de fin
    getTime(h,m,s,c);
    Total_sec_fin := h*3600 + m*60 + s + c/100.0;
    temps[10] += Total_sec_fin - Total_sec_depart;
end;   // Boucle sur g qui varie de 1 à f
```

```
    // Calcul du temps moyen
    for i := 1 to 10 do
        temps[i] := temps[i] / f;


    // Calcul des coefficients
    for i:=1 to 5 do
        if temps[i+5] > 0 then
            coeff[i] := temps[i]/temps[i+5]
        else
            coeff[i] := 0;

    //Affichage des temps (voir le résultat en page 224)
    clrscr;
    write(chr(218));
    for i:=1 to 20 do
        write(chr(196));
    write(chr(194));
    for i:=1 to 9 do
        write(chr(196));
    write(chr(194));
    for i:=1 to 9 do
        write(chr(196));
    write(chr(194));
    for i:=1 to 9 do
        write(chr(196));
    writeln(chr(191));

    writeln(
    chr(179),' Tri             ',chr(179),' Reels   ',chr(179),' Entiers ',chr(179),' k       ',chr(179));

    write(chr(195));
    for i:=1 to 20 do
        write(chr(196));
    write(chr(197));
    for i:=1 to 9 do
        write(chr(196));
    write(chr(197));
    for i:=1 to 9 do
        write(chr(196));
    write(chr(197));
    for i:=1 to 9 do
        write(chr(196));
    writeln(chr(180));

    writeln(
     chr(179),' Tri Bulle ',chr(179),temps[1] :8 :2,' ',chr(179),temps[6] :8 :2,' ',chr(179),coeff[1] :8 :2,' ',chr(179));
    writeln(
     chr(179),' Tri sel. min.',chr(179),temps[2] :8 :2,' ',chr(179),temps[7] :8 :2,' ',chr(179),coeff[2] :8 :2,' ',chr(179));
    writeln(
     chr(179),' Tri sel. min. Max. ',chr(179),temps[3] :8 :2,' ',chr(179),temps[8] :8 :2,' ',chr(179),coeff[3] :8 :2,'
',chr(179));
    writeln(
     chr(179),' Tri par insertion ',chr(179),temps[4] :8 :2,' ',chr(179),temps[9] :8 :2,' ',chr(179),coeff[4] :8 :2,'
',chr(179));
    writeln(
```

```
chr(179),' Tri Shell ',chr(179),temps[5] :8 :2,' ',chr(179),temps[10] :8 :2,' ',chr(179),coeff[5] :8 :2,' ',chr(179) ) ;

    write(chr(192));
    for i:=1 to 20 do
        write(chr(196));
    write(chr(193));
    for i:=1 to 9 do
        write(chr(196));
    write(chr(193));
    for i:=1 to 9 do
        write(chr(196));
    write(chr(193));
    for i:=1 to 9 do
        write(chr(196));
    writeln(chr(217));

    readln;

end.
```

Voici ce qu'on obtient avec 50000 éléments [1] :

- *Reels* est le temps obtenu pour trier un tableau de réels ;
- *Entiers* est le temps obtenu pour trier un tableau d'entiers ;
- *k* est le rapport entre les temps de tris de réels par rapport à celui d'entiers.

Tri	Reels	Entiers	k
Tri Bulle	12.08	9.62	1.26
Tri sel. min.	3.43	3.27	1.05
Tri sel. min. Max.	2.47	2.43	1.02
Tri par insertion	2.62	2.37	1.10
Tri Shell	0.02	0.01	2.33

1. Les temps dépendent, évidemment, de la machine utilisée

2.2 Les sous-programmes

1. Écrire un sous-programme qui calcule $1 + 2 + 3 + \cdots + N$.

```
program Exercice1SsProgramme;

function somme (nombre : integer) : integer;
var i : integer;
   som : integer;
begin
   som := 0;

   for i := 1 to nombre do
      som += i;

   somme := som;   // ou Result := som;
 end;

// Programme principal
var n : integer;
begin
   write('Entrez un nombre : ');
   readln(n);

   writeln('La somme est : ', somme(n));   // Appel de la fonction

   readln;
end.
```

2. Écrire un sous-programme qui teste si un entier est premier.

```
program Exercice2SsProgramme;

function estPremier(nombre : integer) : boolean;
var compteur : integer; // Compte des diviseurs
   i : integer;
begin
   compteur := 0;

   for i := 1 to nombre do
      if nombre mod i = 0 then    // C'est un diviseur
         compteur += 1;

   if compteur = 2 then
      estPremier := true
   else
      estPremier := false;
end;
```

```
// Programme principal
var n : integer;
begin
   write('Entrez un nombre : ');
   readln(n);

   if estPremier(n) = true then   // Appel de la fonction
      writeln('Le nombre est premier')
   else
      writeln('Le nombre n''est pas premier');

   readln;
end.
```

3. Écrire un sous-programme qui calcule a^n (a est un double ; n est un entier positif).

```
program exerciceSsProgramme;

function exposant(a : real; n : integer) : real;
var i : integer;
   produit : real;
begin
   produit := 1;
   for i := 1 to n do
      produit *= a;

   result := produit;    // Ou exposant := produit
end;

var base : real;
   expos : integer;

begin
   write('Entrez un nombre : ');
   readln(base);
   write('Entrez l''exposant de ce nombre : ');
   readln(expos);

   writeln;
   write('La reponse est ',exposant(base, expos):0:2);

   readln;
end.
```

4. Écrire un sous-programme qui calcule a^n (a est un double ; n est un entier quelconque positif, négatif ou nul).

```
program exerciceSsProgramme;

function exposant(a : real; n : integer) : real;
var i : integer;
   produit : real;
begin
   if n = 0 then
   begin
      if a = 0 then
         Result := -1E99    // Valeur "spéciale"
      else
         Result := 1;
   end
   else
   begin
      produit := 1;
      for i := 1 to abs(n) do // Valeur absolue
         produit := produit * a;

      if n < 0 then
         Result := 1 / produit
      else
         Result := produit;
   end;
end;

var base : real;
   expos : integer;

begin
   write('Entrez un nombre : ');
   readln(base);
   write('Entrez l''exposant de ce nombre : ');
   readln(expos);

   writeln;
   write('La reponse est ',exposant(base, expos):0:2);

   readln;
end.
```

5. Écrire un sous-programme qui calcule le pgcd (plus grand commun diviseur) de deux nombres entiers positifs.

```
program exerciceSsProgramme;

function PGCD(A,B : integer) : integer;
var Min, Diviseur : integer;
   trouve : boolean;
begin
   // Quel est le plus petit nombre ?
   if A > B then
      Min := A
   else
      Min := B;

   Trouve := false;
   Diviseur := Min;   // Le PGCD est au plus le plus petit nombre

   // On diminue jusqu'à trouver un diviseur commun
   repeat
      if (A mod Diviseur = 0) and (B mod Diviseur = 0) then
         trouve := true
      else
         Diviseur := Diviseur - 1;
   until trouve;

   Result := Diviseur;
end;

var n1, n2 : integer;
begin
   write('Premier nombre : ');
   readln(n1);
   write('Second nombre : ');
   readln(n2);

   writeln;
   write('Le pgcd de ces deux nombres est ',PGCD(n1,n2));

   readln;
end.
```

6. Écrire un sous-programme qui réduit une fraction à sa plus simple expression.

> **Remarque :** Cet exercice utilise une **procédure** et cette procédure utilise une **fonction**.

```
program reduireSsProgramme;
// ..... Fonction PGCD .....
function PGCD(A,B : integer) : integer;
var Diviseur : integer;
begin
    // Le PGCD est au maximum le plus petit des deux nombres.
    if A > B then
        Diviseur := B
    else
        Diviseur := A;

    // On retire 1 tant qu'au moins un deux nombres de départ n'est pas
    // multiple de Diviseur
    // Utilisation d'une boucle while plutôt que repeat
    while (A mod Diviseur <> 0) or (B mod Diviseur <> 0) do
        Diviseur := Diviseur - 1;

    // Diviseur est le résultat désiré
    Result := Diviseur;
end;

// ..... Procedure Simplification .....
// Réduire une fraction à sa plus simple expression.
// Attention : VAR N2,D2 car ce sont deux résultats mais
// N1,D1 sans var car ce sont des paramètres d'entrée
procedure Simplification(N1,D1:integer; var N2,D2:integer);
begin
    // La procédure se sert de la fonction PGCD (elle l'appelle deux fois)
    N2 := N1 div PGCD(N1,D1);
    D2 := D1 div PGCD(N1,D1);
end;

// ..... Programme principal .....
var N1,D1,N2,D2 : integer;
begin
    write('Numerateur : ');
    readln(n1);
    write('Denominateur : ');
    readln(d1);

    // Appel de la procédure
    Simplification(n1,d1,n2,d2);

    writeln;
    write(n1,'/',d1,' = ',n2,'/',d2);

    readln;
end.
```

7. Écrire un sous-programme qui affiche tous les nombres de Armstrong.

```
program afficheArmstrong;

procedure Armstrong; // Il n'y a pas de paramètre
var nomb : integer;
    reste : integer;
    centaine, dizaine, unite : integer;
begin
    // De part la définition d'un nombre d'Armstrong, la procédure sait qu'il y a trois chiffres
    for nomb := 100 to 999 do
    begin
        // Recherche le chiffre des centaines, des dizaines et des unités
        centaine := nomb div 100;
        reste := nomb mod 100;
        dizaine :=  reste div 10;
        unite := reste mod 10;

        // Armstrong ?
        if nomb = centaine*centaine*centaine + dizaine*dizaine*dizaine +
                  unite*unite*unite then
            writeln(nomb,' est un nombre d''Armstrong');
    end;
end;

begin
    // Appel de la procédure
    Armstrong;

    readln;
end.
```

8. Écrire un sous-programme qui affiche tous les nombres de Pythagore compris entre 1 et N

```
program pythagore;

procedure pytha(n : integer);
var i,j,k : integer;
begin

    for i := 1 to n do
        for j := 1 to n do
            for k := 1 to n do
                if i*i + j*j = k*k then
                    writeln(i,' + ',j,' = ',k,'');
end;

var nb:integer;
begin
    write('Jusqu''ou chercher les nombres de Pythagore ? ');
    readln(nb);

    pytha(nb);

    readln;
end.
```

9. Écrire un sous-programme qui calcule le nombre de voyelles d'une chaîne de caractères.

```
program calculerNombreVoyelles;

function nbvoyelles(chaine : string) : integer;
var nbchar : integer;    // Nombre de caractères de la chaîne
   i : integer;
   compteur : integer;
begin
   nbchar := Length(chaine);
   compteur := 0;

   for i := 1 to nbchar do
      if (chaine[i] = 'a') or (chaine[i] = 'e') or (chaine[i] = 'i') or
         (chaine[i] = 'o') or (chaine[i] = 'u') or (chaine[i] = 'y') then
            compteur += 1;

   Result := compteur;
end;

var chaine:string;
begin
   write('Entrez la chaine de caracteres : ');
   readln(chaine);

   writeln('Le nombre de voyelles dans la chaine = ', nbvoyelles(chaine));
   readln;
end.
```

10. Écrire un sous-programme qui retourne une chaîne de caractères en majuscules. Par exemple, 'oiseau' donne 'UAESIO'.

```
program reverseChaine;

function reverseMaj(ch : string) : string;
var maj : string;
   i : integer;
begin
   maj := '';  // Chaine vide

   for i  := Length(ch) downto 1 do
      maj := maj + ch[i];
   maj := Upcase (maj);

   Result := maj;
 end;

var chaine:string;
begin
   write('Entrez la chaine de caracteres : ');
   readln(chaine);

   writeln('La chaine de caractere en majuscule = ', reverseMaj(chaine));
   readln;
end.
```

11. Écrire un sous-programme qui calcule le produit scalaire de deux tableaux.

$$T1 \bullet T2 = \sum_{i=1}^{n} T1[i]T2[i]$$

```
program produitScalaire;
type tableau = array[1..1000] of integer;

function scalaire(t1, t2 : tableau; taille : integer) : integer;
var i : integer;
   resultat : integer;

begin
   resultat := 0;
   for i := 1 to taille do
      resultat := resultat + t1[i]*t2[i];

   Result := resultat;
end;

var tab1 : tableau;
   tab2 : tableau;
   n : integer;
   i : integer;

begin
   Randomize;

   write('Combien de valeurs par tableau ? ');
   readln(n);

   // Génération aléatoire
   for i := 1 to n do
      tab1[i] := 1 + random(11);
   for i := 1 to n do
      tab2[i] := 1 + random(11);

   // Affichage des deux tableaux
   for i:=1 to n do
      write(tab1[i],' ');
   writeln;
   for i := 1 to n do
      write(tab2[i],' ');

   // Calcul et affichage du résultat
   writeln;
   writeln('Le produit scalaire de ces nombres est ',
      scalaire(tab1,tab2,n) );

   readln;
end.
```

12. Écrire un sous-programme qui calcule le minimum des éléments d'un tableau de réels (double).

```
program minimumTableau;
const nMax = 1000;
type tableau = array[1..nMax] of real;

function minimum(t:tableau; taille:integer) : real;
var i : integer;
   mini : real;
begin
   mini := t[1];

   for i := 2 to taille do
      if t[i] < mini then
         mini := t[i];

    Result := mini;
end;

var tab : tableau;
   i : integer;
   n : integer;

begin
   repeat
      write('Combien de valeurs dans le tableau ? ');
      readln(n);
   until (n > 0) and (n <= nMax);

   for i:=1 to n do
      tab[i] := random * 100;

   for i := 1 to n do
   begin
      write(tab[i]:0:2,' ');
      if i mod 8 = 0 then
         writeln;
   end;

   writeln;
   writeln('Le minimum du tableau est : ', minimum(tab,n):0:2);

   readln;
end.
```

13. Écrire un sous-programme qui génère un tableau d'entiers.

    (a) Avec une fonction.

```
program genereTab;
const nMax = 1000;
type tableau = array[1..nMax] of integer;

function genere(taille : integer) : tableau;
var i : integer;
   tab : tableau;
begin
   Randomize;

   for i := 1 to taille do
      tab[i] := 1 + random(100);

   Result := tab;
end;

var i,n : integer;
   t : tableau;
begin
   write('Taille du tableau ? ');
   readln(n);

   t := genere(n);

   for i := 1 to n do
   begin
      write(t[i],' ');
      if i mod 8 = 0 then
         writeln;
   end;

   readln;
end.
```

(b) Avec une procédure.

```
program genereTab;
const nMax = 1000;
type tableau = array[1..nMax] of integer;

// Attention : VAR (passage par adresse)
procedure genere(var tab : tableau; taille : integer);
var i : integer;
begin
   Randomize;

   for i := 1 to taille do
      tab[i] := 1 + random(100);
end;

var i,n : integer;
   t : tableau;
begin
   write('Taille du tableau ? ');
   readln(n);

   genere(t,n);

   for i := 1 to n do
   begin
      write(t[i],' ');
      if i mod 8 = 0 then
         writeln;
   end;

   readln;
end.
```

14. Écrire la recherche dichotomique grâce à une fonction ; le tableau sera au préalable trié avec une procédure.

```
program triTableauRechercheDichotmique;
const nMax = 1000;
type tableau = array[1..nMax] of integer;

procedure tri(var t:tableau; taille:integer); // Utilisation du tri par insertion
var i,j : integer;
   insere : integer;
begin
   for i:=2 to taille do
   begin
      insere := t[i];
      j := i-1;
      while (j >= 1) and (t[j] >= insere) do
      begin
         t[j+1] := t[j];
         j -= 1;
      end;
      t[j+1] := insere;
   end;
end;
```

```
(*
   La fonction qui suit recherche un élément dans le tableau en utilisant
   la recherche dichotomique. Elle retourne
   -1 si l'élément n'est pas le tableau,
   la place de l'élément s'il y est.
*)
function dichosearch(t:tableau; taille:integer; cle:integer) : integer;
var debut,fin,milieu : integer;
   trouve : boolean;
begin
   debut := 1;
   fin := taille;
   trouve := false;

   while (debut <= fin) and (trouve = false) do
   begin
      milieu := (debut+fin) div 2;
      if t[milieu] = cle then
         trouve := true
      else
         if t[milieu] < cle then
            debut := milieu + 1
         else
            fin := milieu - 1;
   end;

   if trouve then // ⟺ if trouve = true
      Result := milieu
   else
      Result := -1;  // cle ne se trouve pas sans le tableau
end;

// Procédure d'affichage
procedure afficheTableau(t:tableau; taille:integer);
var i : integer;
begin
   for i := 1 to taille do
   begin
      write(t[i],' ');
      if i mod 8 = 0 then
         writeln;
   end;
end;
```

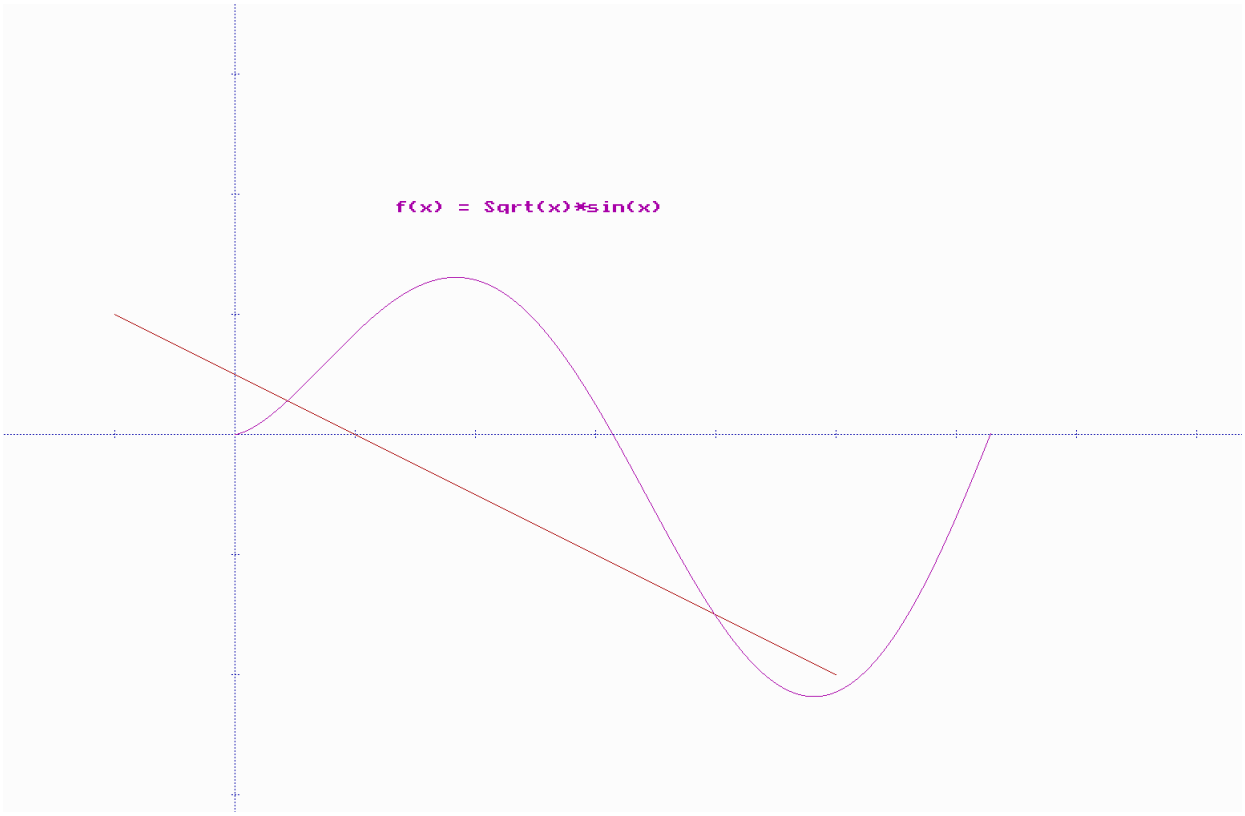
```
// Programme principal
var tab : tableau;
    i : integer;
    n : integer;
    search : integer; // Élément à rechercher
    ou : integer;  // Résultat de la recherche
begin
    Randomize;

    repeat
        write('Combien de valeurs dans le tableau ? ');
        readln(n);
    until (n > 0) and (n <= nMax);

    for i:=1 to n do
        tab[i] := 1 + random(100);

    // Affichage du tableau en appelant la procédure afficheTableau
    writeln;
    affichetableau(tab,n);

    // Appel de la procédure de tri
    tri(tab,n);

    // Réaffichage du tableau (trié)
    writeln;
    writeln;
    afficheTableau(tab,n);

    writeln;
    writeln;
    writeln('Quel nombre recherchez-vous ? ');
    readln(search);

    ou := dichosearch(tab,n,search);
    if  ou <> -1 then
        writeln(search,' a ete trouve a la place ', ou)
    else
        writeln(search,' n''a pas ete trouve dans le tableau');

    readln;
end.
```

15. L'aire d'un cercle de rayon 1 est π et l'aire d'un carré qui contient exactement ce cercle est 4. Ainsi, si un grand nombre de points sont choisis aléatoirement dans la partie supérieure droite du carré, la proportion de ces points qui seront dans le cercle est approximativement $\pi/4$. Calculer une valeur approximative de π.
Pour rappel, l'équation du cercle de rayon 1 est $x^2 + y^2 = 1$.

```
program CalculPI;

// La fonction exécute la génération aléatoire n fois
function approchePI(n:integer) : real;
var i : integer;
   x,y : real; // Nombres générés aléatoirement entre 0 et 1
   compteur : integer;  // Nombre de fois que l'expérience est à l'intérieur du cercle
begin
   compteur := 0;

   for i:=1 to n do
   begin
      x := random;     // x ∈ [0,1[
      y := random;     // y ∈ [0,1[
      if x*x + y*y <= 1 then   // Tombe dans le cercle
         compteur := compteur + 1;
   end;

   Result := compteur/n * 4;
end;

begin
   write(approchePI(1000000):0:8);   // 1000000 de fois l'expérience

   readln;
end.
```

2.3 Graphisme

Réaliser l'exercice suivant :




```
program Graphe;
uses Graph,crt;
var pilote, mode : smallint;
    i : integer;
    Largeur,      // Largeur de la fenêtre graphique
    Hauteur,      // Hauteur de la fenêtre graphique
    Echelle,      // Une unité mathématique correspond à Echelle pixels
    MilieuX, milieuY : integer;     // Origine (0,0) mathématique
    pas : Real;
    xMath, yMath : Real; // Coordonnées mathématiques
    xPixels, yPixels : Integer;     // Coordonnées en pixels
    x1, x2 , y1, y2 : Real;// Coordonnées de deux points (peuvent avoir des décimales)

// La fonction à tracer
function f(x:real) : real;
begin
    Result := Sqrt(x) * sin(x);
end;
```

```
begin
    InitGraph(pilote,mode,'');// Initialisation de la fenêtre graphique

    // Initialisation des variables
    Echelle := 150;
    Largeur := GetMaxX;
    Hauteur := GetMaxY;
    MilieuX := 2 * Echelle;
    MilieuY := Hauteur div 2;

    // Mettre un fond blanc (plus lisible qu'un fond noir)
    SetBkColor(White);
    ClearViewPort;

    // Définition du style des axes
    SetLineStyle(DottedLn, 0, NormWidth);// Pointillés
    SetColor(Blue); // Les axes seront en bleu

    // Axe des X
    MoveTo(0,MilieuY);
    LineTo(Largeur,MilieuY);
    // Axe des Y
    MoveTo(MilieuX, 0);
    LineTo(MilieuX, hauteur);
    // Graduations de l'axe des X (10 pixels de haut)
    // Gauche de l'axe
    i := 0;
    repeat
        MoveTo(MilieuX-i, MilieuY-5);
        LineTo(MilieuX-i, MilieuY+5);
        i := i + echelle;
    until i >= 2*Echelle;
    // Droite de l'axe
    i := 0;
    repeat
        MoveTo(MilieuX+i, MilieuY-5);
        LineTo(MilieuX+i, MilieuY+5);
        i := i + echelle;
    until i >= largeur;

    // Graduations sur l'axe des Y (10 pixels de large)
    i := 0;
    repeat
        MoveTo(MilieuX-5, MilieuY-i);
        LineTo(MilieuX+5, MilieuY-i);
        MoveTo(MilieuX-5, MilieuY+i);
        LineTo(MilieuX+5, MilieuY+i);
        i := i + echelle;
    until i > hauteur div 2;
```

```
    // Tracé d'un segment de droite entre deux points dont les coordonnées
    // seront lues à l'écran
    write ('Abcisse du premier point : ');
    readln (x1);
    write ('Ordonnee du premier point : ');
    readln (y1);
    write ('Abcisse du deuxieme point : ');
    readln (x2);
    write ('Ordonnee du deuxieme point : ');
    readln (y2);
    SetColor (Red);
    SetLineStyle(SolidLn, 0, NormWidth);// Continu
    MoveTo (milieuX + Round(x1*Echelle), milieuY - Round(y1*Echelle));
    LineTo (milieuX + Round(x2*Echelle), milieuY - Round(y2*Echelle));

    // Tracé de f(x) entre 0 et 2π
    SetLineStyle(SolidLn, 0, NormWidth);// Pointillés
    SetColor(Magenta);
    pas := 1/Echelle;
    xMath := 0;
    yMath := f(0); // Vaut 0
    xPixels := milieuX + Round(xMath*Echelle);
    yPixels := milieuY - Round(yMath*Echelle);
    MoveTo(xPixels,yPixels); // Bien se positionner au départ

    repeat
       xMath := xMath + pas;
       yMath := f(xMath);
       xPixels := milieuX + Round(xMath*Echelle);
       yPixels := milieuY - Round(yMath*Echelle);
       LineTo(xPixels,yPixels);
    until xMath > 2 * pi;

    // Ecrit le texte f(x) = sqrt(x)*sin(x)
    SetTextStyle(GothicFont,HorizDir,2);
    OutTextXY(MilieuX + 200,250,'f(x) = Sqrt(x)*sin(x)');

    readln;
    CloseGraph;
end.
```

2.4 Les fichiers texte

1. Compter le nombre de lignes d'un fichier texte.

```
program ExFichiers;
var fich : text;
    ligne : string;
    compteur : integer;
begin
    Assign(fich, 'C:\Exercices\fichier.txt');
    Reset(fich);

    compteur := 0;
    while not Eof(fich) do
    begin
        Readln(fich, ligne);
        compteur := compteur + 1;
    end;

    Close(fich);
    writeln;
    writeln('Il y a ',compteur,' lignes dans le texte');
    readln;
end.
```

2. Mettre tous les caractères d'un fichier texte en majuscules (dans un autre fichier).

```
program ExFichiers;
var fichIn, fichOut : text;
    ligne : string;
begin
    Assign(fichIn, 'C:\fpc\bdlog.txt');
    Reset(fichIn); // Ouverture en lecture
    Assign(fichOut, 'C:\fpc\maj.txt');
    Rewrite(fichOut); // Création (ou effacement) du fichier

    while not Eof(fichIn) do
    begin
        Readln(fichIn, ligne);
        Writeln(fichOut, upCase(ligne));
    end;

    Close(fichIn);
    Close(fichOut);
end.
```

3. Calculer la fréquence d'apparition des lettres dans un fichier texte.

```
program frequenceLettres;
var fich : text;

    // Tableau qui permet de compter le nombre de fois que chaque lettre apparaît.
    // tab['A'] contient le nombre de 'A', ..., tab['Z'] contient le nombre de 'Z'.
    tab : array['A'..'Z'] of integer;
    tabFrequence : array['A'..'Z'] of real;

    i : integer;
    ligne : string;
    j : char;    // Pour les boucles sur le tableau de char (tab)
    compteurLettres : integer; // Nombre de lettres
begin
    for j := 'A' to 'Z' do
       tab[j]:=0;
    compteurLettres:=0;

    Assign(fich, 'C:\fpc\bdlog.txt');
    Reset(fich);

    while not Eof(fich) do
    begin
       Readln(fich, ligne);
       ligne := Upcase(ligne); // Mise en majuscules
       // Parcourir tous les caractères de la chaîne
       for i := 1 to Length(ligne) do
       begin
          // Ne tenir compte que des lettres
          if (ligne[i] >= 'A') and (ligne[i] <= 'Z') then
          begin
             tab[ligne[i]] += 1;  // Le cœur de l'exercice
             compteurLettres += 1;
          end;
       end;
    end;

    Close(fich);

    writeln;
    writeln('Il y a ',compteurLettres,' lettres dans le fichier');
    writeln;

    // Transformer tab en tableau des fréquences (pourcentage)
    for j := 'A' to 'Z' do
       tabFrequence[j] := tab[j]/compteurLettres * 100;

    for j := 'A' to 'Z' do
       writeln(j,' : ',tabFrequence[j]:0:1,' %');

    readln;
end.
```

4. Compter le nombre de mots d'un fichier texte.

```
(*
    Compte le nombre de mots d'un fichier en supposant qu'il n'y ai pas de
    lettres accentuées.
    Cette programme utilise les ensembles mais pourrait s'en passer
*)
program CompteurMots;
var fichier: Text;
    nMots: integer;
    ligne: string;
    i: integer;
    nouveauMot : boolean;
begin
   // Ouverture
   Assign(fichier, 'C:\fpc\bdlog.txt');
   Reset(fichier);

   // Comptage
   nMots := 0;

   while not eof(fichier) do
   begin
      Readln(fichier,ligne);
      nouveauMot := False;
      for i := 1 to Length(ligne) do
      begin
         // Pour chaque caractère, on regarde si c'est une lettre
         if (ligne[i] IN ['A'..'Z','a'..'z']) and (nouveauMot = False) then
         begin
            inc(nMots); // Si on n'est pas déjà entrain de traiter un mot,
                        // on augmente le compteur de mots
            nouveauMot := True;// On est entrain de traiter un nouveau mot
         end
         else
            if not (ligne[i] IN ['A'..'Z','a'..'z']) then // Pas une lettre
               nouveauMot := False;
      end;
   end;

   Close(fichier); // Fermeture du fichier

   writeln('Il y a ',nMots,' mots dans le fichier');

   readln;
end;
```

5. Lire un fichier texte et y remplacer les voyelles par #.

Première solution

```
program ExFichiers;
var fichIn, fichOut : text;
    ligne : string;
    i : integer;
    car : char; // ie caractère de la ligne en majuscule
begin
    Assign(fichIn, 'C:\fpc\bdlog.txt');
    Reset(fichIn); // Ouverture en lecture
    Assign(fichOut, 'C:\fpc\voyelles.txt');
    Rewrite(fichOut); // Création (ou effacement) du fichier

    while not Eof(fichIn) do
    begin
        Readln(fichIn, ligne);
        // Traitement de la ligne
        for i := 1 to Length(ligne) do
        begin
            // Mettre le caractère (ie)en majuscule dans une autre variable
            // pour ne pas changer la valeur du caractère.
            car := upCase(ligne[i]);
            if (car = 'A') or (car = 'E') or (car = 'I') or
               (car = 'O') or (car = 'U') or (car = 'Y') then
                  ligne[i] := '#';
        end;
        Writeln(fichOut, ligne);
    end;

    Close(fichIn); Close(fichOut);
end.
```

Deuxième solution (utilisation de Pos qui indique la position d'une sous-chaîne dans une chaîne).

```
program ExFichiers;
var fichIn, fichOut : text;
    ligne : string;
    i : integer;
    voyelles : string;    // Contiendra toutes les voyelles
begin
    Assign(fichIn, 'C:\fpc\UControle.pas');    Reset(fichIn);
    Assign(fichOut, 'C:\fpc\voyelless.txt');   Rewrite(fichOut);

    voyelles := 'aeiouyAEIOUY';

    while not Eof(fichIn) do
    begin
        Readln(fichIn, ligne);
        for i := 1 to Length(ligne) do    // Traitement de la ligne
            if Pos(ligne[i], voyelles) > 0 then // C'est une voyelle
                ligne[i] := '#';
        Writeln(fichOut, ligne);
    end;

    Close(fichIn); Close(fichOut);
end.
```

6. Dans un fichier HTML, remplacer les balises gras par italique.
   - devient
   - devient

```
program ExFichiers;
var fichIn : text;
    fichOut : text;
    ligne : string;
    i : integer;
begin
    Assign(fichIn, 'C:\fpc\essai.html');
    Assign(fichOut, 'C:\fpc\resultat.html');

    Reset(fichIn);
    Rewrite(fichOut);

    while not Eof(fichIn) do
    begin
        Readln(fichIn, ligne);

        // Remplacer <b> par <i>
        for i := 1 to Length(ligne)-2 do
        begin
            if ligne[i]+ligne[i+1]+ligne[i+2] = '<b>' then
            begin
                ligne[i] := '<';     // Inutile
                ligne[i+1] := 'i';
                ligne[i+2] := '>';
            end;
        end;

        // Remplacer </b> par </i>
        for i := 1 to Length(ligne)-3 do
        begin
            if ligne[i]+ligne[i+1]+ligne[i+2]+ligne[i+3] = '</b>' then
            begin
                ligne[i] := '<';
                ligne[i+1] := '/';
                ligne[i+2] := 'i';
                ligne[i+3] := '>';
            end;
        end;

        // Écriture dans le fichier
        Writeln(fichOut,ligne);
    end;

    Close(fichIn);
    Close(fichOut);

    write('Travail termine!');
    readln;
end.
```

7. Compter le nombre de fois qu'un mot apparaît dans un fichier.

```
program CompterMot;
var fichier_Texte : Text;
    ligne : string;
    motACompter : string;
    compteur : integer;  // Compte combien de fois le mot est dans le fichier
    i : integer;
begin
    Assign(fichier_Texte,'C:\fichier.pas');
    Reset(fichier_Texte);

    // Quel mot recherche-t-on ?
    write('Quel mot ?');
    readln(motACompter);

    compteur := 0;
    while not Eof(fichier_Texte) do
    begin
        Readln(fichier_Texte, ligne); // Lire une ligne du fichier
        // Tester si le mot se trouve à la ie position
        // Le programme extrait à partir du ie caractère autant de caractères
        // que la longueur du mot
        for i := 1 to Length(ligne) - Length(motACompter) + 1 do
            if Copy(ligne,i, Length(motACompter)) = motACompter then
                compteur := compteur + 1;
    end;

    Close(fichier_Texte);

    writeln(compteur);

    readln;
end.
```

2.5 Les fichiers d'enregistrements

Gestion "simplifiée" d'un fichier d'élèves (cf. exercice page 127)

```
program FichierEleves;
uses sysutils,crt;

type TEleve =
      record
         numero:integer;
         nom:string;
         prenom:string;
         sexe:char;
         nat:string;
         classe:string;
         localite:string;
         points:integer;
      end;
    fichierEleves = file of TEleve ;

(************************** Affiche tous les élèves **************************)
procedure showAll(chemin:string) ;
var unEleve:TEleve;
    fich:fichierEleves;
begin
   // Quitte la procédure si le fichier n'existe pas
   if not fileExists(chemin) then
   begin
      write('Fichier inexistant!');
      readln;
      exit;
   end;

   clrscr;  // Efface l'écran
   assign(fich,chemin);
   reset(fich);

   while not eof(fich) do
   begin
      read(fich,unEleve);
      writeln(unEleve.numero,' ',
         unEleve.nom,StringOfChar(' ',20-length(unEleve.nom)),
         unEleve.prenom,StringOfChar(' ',20-length(unEleve.prenom)),
         unEleve.sexe,StringOfChar(' ',2-length(unEleve.sexe)),
         unEleve.nat,StringOfChar(' ',5-length(unEleve.nat)),
         unEleve.classe,StringOfChar(' ',5-length(unEleve.classe)),
         unEleve.localite,StringOfChar(' ',25-length(unEleve.localite)),
         unEleve.points,' ');
   end;
   close(fich);

   readln;
end;
```

```
(***************** Recherche les élèves connaissant un nom ****************)
procedure searchName(chemin:string; nom:string);
var unEleve:TEleve;
    fich:fichierEleves;
begin
   // Quitte la procédure si le fichier n'existe pas
   if not fileExists(chemin) then
   begin
      write('Fichier inexistant!');
      readln;
      exit;
   end;

   clrscr;
   assign(fich,chemin);
   reset(fich);

   while not eof(fich) do
   begin
      read(fich,unEleve);
      if unEleve.nom = nom then
      begin
         writeln(unEleve.numero,' ',
            unEleve.nom,StringOfChar(' ',20-length(unEleve.nom)),
            unEleve.prenom,StringOfChar(' ',20-length(unEleve.prenom)),
            unEleve.sexe,StringOfChar(' ',2-length(unEleve.sexe)),
            unEleve.nat,StringOfChar(' ',5-length(unEleve.nat)),
            unEleve.classe,StringOfChar(' ',5-length(unEleve.classe)),
            unEleve.localite,StringOfChar(' ',25-length(unEleve.localite)),
            unEleve.points,' ');
       end;
   end;
   close(fich);

   readln;
end;

(***************** Recherche l'élève connaissant son numéro ****************)
procedure searchNum(chemin:string; num:integer);
var unEleve:TEleve;
    fich:fichierEleves;
begin
   // Quitte la procédure si le fichier n'existe pas
   if not fileExists(chemin) then
   begin
      write('Fichier inexistant!');
      readln;
      exit;
   end;

   clrscr;
   assign(fich,chemin);
   reset(fich);
```

```
    while not eof(fich) do
    begin
        read(fich,unEleve);
        if unEleve.numero = num then
        begin
          writeln(unEleve.numero,' ',
              unEleve.nom,StringOfChar(' ',20-length(unEleve.nom)),
              unEleve.prenom,StringOfChar(' ',20-length(unEleve.prenom)),
              unEleve.sexe,StringOfChar(' ',2-length(unEleve.sexe)),
              unEleve.nat,StringOfChar(' ',5-length(unEleve.nat)),
              unEleve.classe,StringOfChar(' ',5-length(unEleve.classe)),
              unEleve.localite,StringOfChar(' ',25-length(unEleve.localite)),
              unEleve.points,' ');
        end;
    end;
    close(fich);

    readln;
end;

(***************** Fonction pour compter le nombre de filles ***************)
function nbrfilles(chemin:string) : integer;
var compteur:integer;
     unEleve:TEleve;
     fich:fichierEleves;
begin
    // Quitte la procédure si le fichier n'existe pas
    if not fileExists(chemin) then
    begin
        write('Fichier inexistant!');
        readln;
        exit;
    end;

    clrscr;
    assign(fich,chemin);
    reset(fich);

    compteur:=0;
    while not eof(fich) do
    begin
        read(fich,unEleve);
        if unEleve.sexe = 'F' then
          compteur:=compteur+1;
    end;
    close(fich);
    Result := compteur;
end;
```

```
(*** Afficher les renseignements de l'élève qui a le plus de points ***)
procedure pointsMax(chemin:string);
var  unEleve:TEleve;
   unEleveMax:TEleve;     // Elève qui a le plus de points
   fich:fichierEleves;
   maxPoints:integer;
begin
   // Quitte la procédure si le fichier n'existe pas
   if not fileExists(chemin) then
   begin
      write('Fichier inexistant!');
      readln;
      exit;
   end;

   assign(fich,chemin);
   reset(fich);

   // Trouver l'élève qui a le plus de points
   maxPoints := 0;
   while not eof(fich) do
   begin
      read(fich,unEleve);
      if unEleve.points > maxPoints then
      begin
         maxPoints := unEleve.points;
         unEleveMax := unEleve;   // Retenir l'élève
      end;
   end;

   // Afficher tous les renseignements de l'élève trouvé précédemment
   clrscr;
   writeln(unEleveMax.numero,' ',
      unEleveMax.nom,StringOfChar(' ',20-length(unEleveMax.nom)),
      unEleveMax.prenom,StringOfChar(' ',20-length(unEleveMax.prenom)),
      unEleveMax.sexe,StringOfChar(' ',2-length(unEleveMax.sexe)),
      unEleveMax.nat,StringOfChar(' ',5-length(unEleveMax.nat)),
      unEleveMax.classe,StringOfChar(' ',5-length(unEleveMax.classe)),
      unEleveMax.localite,StringOfChar(' ',25-length(unEleveMax.localite)),
      unEleveMax.points,' ');
   close(fich);

   readln;
end;
```

```
(************* Suppression d'un élève connaissant son numéro ***************)
procedure supEleve(chemin:string;num:integer);
var unEleve:TEleve;
   fich,fich2:fichierEleves;
   num:integer;
begin
   // Quitte la procédure si le fichier n'existe pas
   if not fileExists(chemin) then
   begin
      write('Fichier inexistant!');
      readln;
      exit;
   end;

   clrscr;
   assign(fich,chemin);
   reset(fich);
   assign(fich2,'c:/test/Eleves2.rec');
   rewrite(fich2) ;

   while not eof(fich) do
   begin
       read(fich,unEleve);
       if  unEleve.numero <> num then
       begin
         write(fich2,unEleve);
       end;
   end;

   close(fich);
   close(fich2);

   deletefile(chemin);
   renamefile('c:/test/Eleves2.rec',chemin);
end;
```

```
(********************** Tri par ordre alphabétique **********************)
procedure triAlphabet(chemin:string;cheminTrie:string);
var t : array[1..1000] of TEleve;
    fich:file of TEleve;
    fichTri:file of TEleve;
    i:integer;
    memoire:TEleve;
     echange:boolean;
begin
   // Quitte la procédure si le fichier n'existe pas
   if not fileExists(chemin) then
   begin
      write('Fichier inexistant!');
      readln;
      exit;
   end;

   clrscr;
   assign(fich,chemin);
   reset(fich);

   // Copier le fichier dans le tableau
   i:=1;
   while not eof(fich) do
   begin
       read(fich,t[i]);
       i:=i+1;
   end;
   close(fich);

   // Tri du tableau (bulle)
   // fileSize(fich) est le nombre d'éléments "intéressants" du tableau
   repeat
      echange := false;
      for i:=1 to fileSize(fich) do
      begin
         if t[i].nom > t[i+1].nom then
         begin
            echange := true;
            memoire := t[i];
            t[i] := t[i+1];
            t[i+1] := memoire;
         end;
      end;
   until echange = false;

   // Copier le tableau dans le fichier trié
   assign(fichTri,cheminTrie);
   rewrite(fichTri);
   for i:=1 to fileSize(fich) do
      write(fichTri,t[i]);

   close(fichTri);
end;
```

```
(************************ Programme principal ************************)
var choix:char;
    chemin:string;
    cheminTrie:string;
    nom:string;
    num:integer;
begin
   chemin:='c:/test/Eleves.rec';
   Repeat
      clrscr;
      writeln('Entrez a pour afficher tous les eleves ');
      writeln('Entrer r pour rechercher un eleves avec le nom ' );
      writeln('Entrer n pour rechercher un eleves avec le numero ' );
      writeln('Entrez f pour compter le nombres de filles ');
      writeln('Entrez m pour voir les eleves qui ont eu le max ');
      writeln('Entrez s pour supprimer un eleves avec son numero') ;
      writeln('Entrez t pour trier par ordre alphabétique ');
      writeln('Entrez q pour quitter le programme ');
      readln(choix);
      case choix of
          'a' : showAll(chemin);
          'r' : begin
                  write('Entrez le nom a rechercher ');
                  readln(nom);
                  nom := upcase(nom);
                  searchName(chemin,nom);
                end;
          'n' : begin
                  write('Entrez le numero de l''eleve a rechercher ');
                  readln(num);
                  searchNum(chemin,num);
                end;
          'f' : begin
                  write('Le nombre de filles vaut ', nbrfilles(chemin));
                  readln;
                end;
          'm' : pointsMax(chemin);
          's' : begin
                  write('Entrez le numero de l''eleve a supprimer ');
                  readln(num);
                  supEleve(chemin,num);
                end;
          't' : begin
                  cheminTrie := 'c:/test/ElevesTrie.rec';
                  triAlphabet(chemin,cheminTrie);
                end;
      end;
   until choix='q' ;
end.
```