INITIATION A TURBO PASCAL

09/05/2003

J.P. CHEVAUX

GENERALITES

La machine, ou plus exactement le microprocesseur, ne peut exécuter que des instructions codées en binaire. Mais l'écriture de ces codes est particulièrement fastidieuse et n'a été pratiquée qu'au tout début de la programmation. Très rapidement, on a utilisé les codes hexadécimaux. Ils n'apportent aucun confort supplémentaire en matière de programmation mais permettent de réduire considérablement les erreurs de frappe.

La nécessité d'associer des instructions littérales aux codes, s'est imposée très rapidement. L'assembleur répond à cette nécessité. Chaque code du microprocesseur est associé à un mnémonique (LDA,MOV...) et les valeurs sont notées en hexadécimal. Les programmes écrits en assembleur ont l'avantage de pouvoir exploiter au maximum les possibilités du microprocesseur et fournissent donc des programmes très rapides.

Cette dernière façon de faire est difficile à mettre en oeuvre et nécessite une grande pratique pour obtenir des résultats satisfaisants. D'autant que la détection d'erreurs de programmation n'est pas chose facile. Néanmoins, à l'heure actuelle, on programme toujours certaines applications en langage machine car on a besoin d'une rapidité très importante pour l'exécution du programme. C'est le cas, en particulier, dans les applications dites 'en temps réel' dans les domaines de l'automatisme et de la robotique. Malgré tout, les performances sans cesse croissantes des matériels tendent à faire disparaître cette programmation.

Les années 50 ont vu apparaître les langages dits "évolués". Ceux ci permettent de s'affranchir des contraintes du langage de la machine et permettent de décrire le but à atteindre et non pas la façon d'y arriver. Par exemple, dans un langage évolué on donnera l'instruction Writeln('Ceci est un texte à afficher') qui donne le but à atteindre: afficher le texte entre apostrophes sur l'écran et placer le curseur sur la ligne suivante. En assembleur ou en langage machine il serait nécessaire de donner la liste de toutes les instructions machines qui permettent d'amener le premier caractère de la chaîne dans le registre x puis de l'afficher, de passer au caractère suivant en ayant pris soin de tester que ce n'est pas la fin de la chaîne de caractère.....etc.

Le premier langage évolué utilisé par les programmeurs fut le BASIC (1958). Ce langage prévu au départ pour l'initiation à la programmation a rapidement envahi le marché de la programmation.

Au début des années 70 le PASCAL et le C apparaissent. Ils ont comme caractéristique de structurer la programmation. C'est à dire de programmer suivant des méthodes rigoureuses et donc efficaces. La structuration des programmes permet en outre une maintenance plus facile des applications.

Au début des années 90 apparaissent des langages dit "orientés objets". La grande nouveauté de ces langages est de proposer au programmeur une approche différente dans l'écriture du programme: on n'écrit plus un programme en décrivant uniquement une suite d'action, mais on construit des objets (par exemple une fenêtre) et on lui attribue des méthodes (par exemple: ouvrir une fenêtre, la fermer, lui adjoindre une barre d'icône...etc).

Il existe un grand nombre de langage sur le marché, sans doute plus de mille. Et il serait vain de vouloir tous les apprendre. Les principes fondamentaux de la programmation que nous vous proposons de découvrir se retrouvent dans tous les langages. Une fois acquis, le passage d'un langage à un autre se fait sans trop de difficulté. Nous avons choisi pour cette initiation de travailler en TurboPascal. Ce n'est pas le langage des développeurs professionnels, qui serait plutot le C. Il oblige à respecter une syntaxe et une structuration très rigoureuses. En contrepartie, Pascal vous donnera toujours un message d'erreur qui vous permettra de ne pas rester bloqué devant un programme qui n'est pas correct.

TurboPascal

TurboPascal est un environnement de développement qui permet d'écrire des programmes en langage structuré de haut niveau. Il intègre à partir de sa version 5.5 le concept de P.O.O.[1].

TurboPascal est un environnement de développement qui réunit un compilateur, un éditeur de texte, un Linker et un débogueur[2].

Mécanisme de compilation

Les langages de programmation ne sont rien d'autres que des traducteurs. Ils lisent un texte (le programme) et font le nécessaire pour que les instructions du programme deviennent compréhensibles par la machine. C'est dans leur façon de traduire qu'ils se différencient. Ils se divisent en deux grandes familles.

Les compilateurs

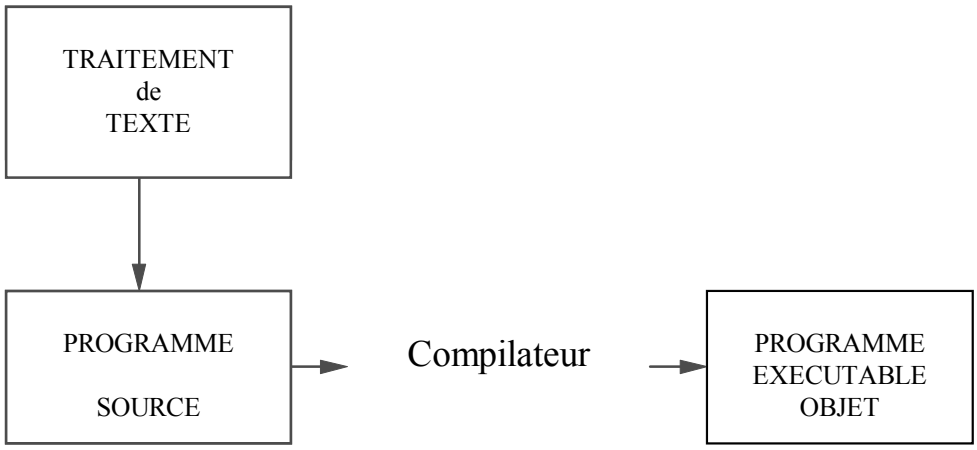
Dans ce cas, le programmeur écrit un texte qui est le PROGRAMME SOURCE. Ce programme est écrit à l'aide d'un éditeur de texte qui est un traitement de texte rudimentaire. Lorsque son programme est entièrement rédigé, le programmeur compile son programme. cette opération consiste à faire lire le programme source par le

[1] Programmation Orientée Objet

[2] Outil de mise au point des programmes (à partir de la version 5.0)

compilateur qui produit un nouveau programme: le PROGRAMME OBJET. Ce dernier programme est entièrement rédigé dans le langage de la machine est pourra être exécuté.

Le compilateur traduit le programme à la manière du traducteur de livres qui s'isole pendant un certain temps pour faire la traduction d'un ouvrage et donnera au lecteur une traduction complète et définitive.

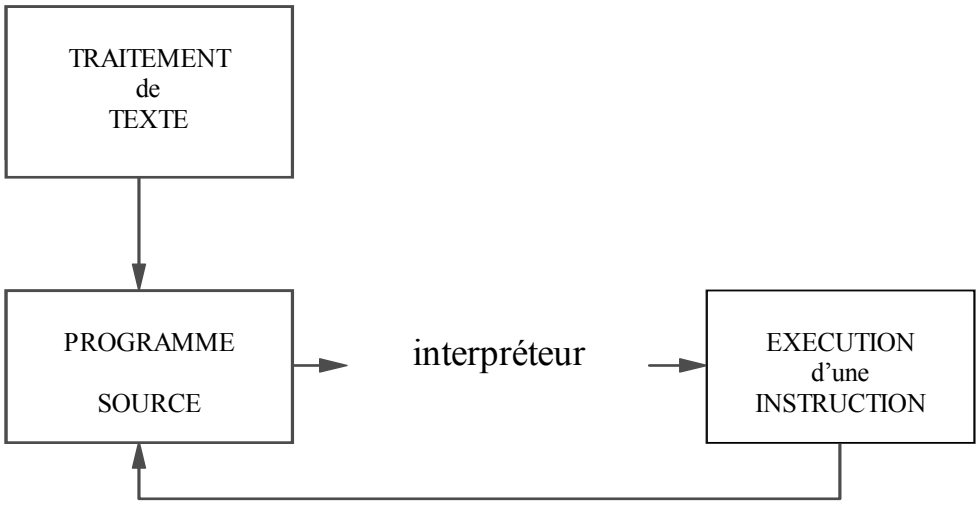


Il faut retenir que seul le programme source est modifiable, en cas d'erreur par exemple, et que seul le programme objet est exécutable. Le programme source est facilement modifiable.

Les interpréteurs

Le programmeur utilise encore une fois l'éditeur de texte pour écrire son programme. Lorsqu'il a terminé, il demande directement l'exécution de son programme. L'interpréteur va agir de la façon suivante. Il lit une première instruction, l'interprète et l'exécute. Il lit ensuite la seconde instruction, l'interprète et l'exécute. Et ainsi de suite jusqu'à la fin du programme. Si l'on demande une nouvelle exécution, l'interpréteur recommencera le même travail.

L'interpréteur agit ainsi à l'image de l'interprète qui traduit un discours. Il n'y a jamais production de programme objet.



L'exécution d'un programme interprété nécessite que L'interpréteur soit présent en mémoire au moment de l'exécution. C'est à dire que, d'une part, toutes les données utilisées par le programme doivent être explicitement déclarées, et d'autre part, que toutes les instructions nécessaires pour obtenir le résultat souhaité soient contenues dans le programme. Autrement dit, TurboPascal ne prendra aucune initiative.

Avantages et inconvénients

Les programmes sont faciles à mettre au point. On peut en particulier consulter l'état des variables après une interruption du programme.

La vitesse d'exécution du programme est considérablement ralentie par le phénomène d'interprétation. On peut pallier cet inconvénient en compilant le programme définitivement mis au point.

Algorithme

La notion d'algorithme est fort ancienne, et l'on peut en donner la définition suivante :

- Spécification d'un ensemble d'instructions, données dans un certain ordre, qui permet de résoudre le problème posé.
- L'algorithme opère sur des données, donne un résultat dans un temps fini à l'aide d'un nombre d'instructions également fini.

L'algorithme se situe donc naturellement entre le problème et le programme. Il permet de décrire la suite d'instructions à enchaîner pour obtenir la résolution d'un problème. Il est indépendant du langage de programmation ainsi que de la machine sur laquelle il sera implémanté.

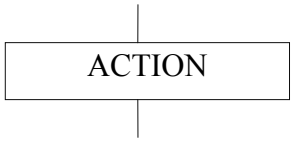
Problème ⟶ Algorithme ⟶ Programme

Représentation d'un algorithme

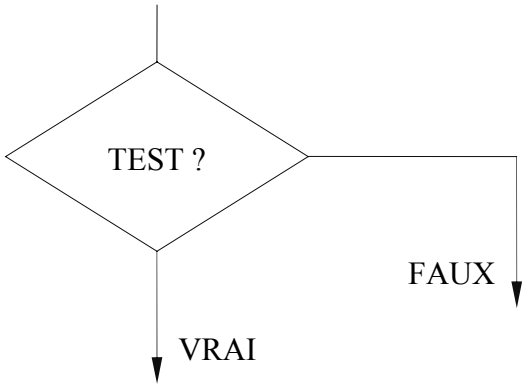
De nombreuses notations existent qui permettent de décrire un algorithme, en voici deux:

Les organigrammes

Les organigrammes sont basés sur une description symbolique des algorithmes. Ils sont très "parlants" et permettent de visualiser rapidement un algorithme simple. En effet, les organigrammes deviennent inextricables dès que l'on veut traiter un algorithme un peu complexe, et il faut impérativement dans ce cas utiliser d'autres outils de description.



Représente une action. modification de variables, lecture, écriture ..



Représente des modifications conditionnelles du déroulement du programme dont le résultat dépend d'une condition. Si le résultat du test de la condition est vrai, on suivra l'organigramme par le chemin logique vrai. Dans la cas contraire, on suivra l'organigramme par le chemin logique faux.

PseudoPascal

Cette façon de représenter un algorithme est très répandue. Elle utilise le Français et les structures de base TANTQUE..REPETE et SI..ALORS..SINON. Un exemple simple sur l'art de faire le café permet de l'illustrer.

```
Indiquer le nombre de tasses désirées
TANTQUE le nombre de tasses est supérieur à 10
   REPETE   Impossible, la cafetière va déborder
            Choisir un nouveau nombre de tasses
Mettre l'eau dans la cafetière
Mettre un filtre
SI le café est en grains
   ALORS    Prendre le moulin à café
            Prendre du café dans le pot à café en grains
            Moudre le café
   SINON    Prendre le pot à café moulu
Remplir une mesure de café
Verser son contenu dans le filtre
TANTQUE le nombre de mesures est insuffisant
   REPETE   Prendre une nouvelle mesure de café
            Verser son contenu dans le filtre
Appuyer sur le bouton de mise en marche.
```

Il est inutile de donner une longue explication. C'est clair!

Remarquez, et c'est fondamental que la description n'utilise que deux structures SI..ALORS..SINON.. et TANTQUE..REPETE.. Ces deux structures permettent de décrire **n'importe quel algorithme.** Quelque soit la complexité d'un problème à résoudre, et à condition qu'il y ait une solution au problème posé, on pourra le décrire à l'aide de ces deux structures. C'est l'art d'agencer ces deux structures qui permet d'élaborer un algorithme. Il suffira ensuite de traduire l'algorithme dans un langage de programmation.

STRUCTURE D'UN PROGRAMME

Généralités

Un programme TurboPascal est structuré en trois parties : l'en-tête, les déclarations et le programme principal. Si l'en-tête et le programme principal sont de tailles réduites, il n'en est pas de même pour la partie des déclarations qui peut atteindre des milliers de lignes puisqu'on déclare aussi bien dans cette zone: les types, les constantes, les variables, les fonctions que les procédures. L'exemple proposé ci-dessous présente une structure minimale.

```
PROGRAM NomduProgramme;
Uses Crt;

Const
      Annee = 1994;
Var
      Age : Integer;
      Nom : String;

Begin
      ClrScr;
      Writeln ('Donner votre nom : ');
      Readln (Nom);
      Writeln ('Donner votre âge : ');
      Readln (Age);
      Writeln ('Bonjour ',Nom);
      Writeln ('Vous êtes né en : ',Annee-Age);
end.
```

EN-TETE

DECLARATIONS
- variables
- constantes
- types
- fonctions
- procédures

PROGRAMME PRINCIPAL

Un programme Pascal se compose de trois parties:

L' en-tête

C'est une partie qui est trés courte dans laquelle le programmeur déclare la liste des unités précompilées (voir Memento) que le programme utilisera. Les unités précompilées contiennent la définition de procédures et de fonctions qui peuvent être définies dans le langage ou par le programmeur. L'unité CRT contient les définitions des procédures et fonctions permettant de gérer l'écran et le clavier. Un certain nombre

d'unités sont disponibles telles que:

UNITE	*Gestion*
CRT	Clavier, Ecran, Son, Couleurs ...
DOS	Date et Heure, Gestion des disques...
PRINTER	Gestion de l'impression
STRINGS	Gestion des chaines AZT
GRAPH	Gestion du graphisme

Les identificateurs

Pour définir le nom du programme ou les noms de variables ou de procédures, on utilise des identificateurs. Un identificateur est une suite de caractères qui ne doit contenir aucun espace et qui commence par une lettre. On peut insérer des caractères de soulignement _ pour aérer l'écriture:

Taxe_sur_la_valeur_ajoutee
TVA
T

sont des identificateurs valides.

Les déclarations

Cette zone permet de déclarer les éléments utilisés dans le programme principal. Variables, Constantes, types ainsi que les fonctions et procédures. Cette partie peut atteindre plusieurs milliers de lignes et constitue l'essentiel d'un programme Pascal. Dans un premier temps nous nous contenterons de déclarer des variables et des constantes. Mais de quoi s'agit-il?

Les variables

Une variable est un espace mémoire qui va permettre de stocker des valeurs. Et comme c'est une variable, la valeur qu'elle contient va pouvoir changer tout au long du déroulement du programme.

Pour prendre une image simple, une variable est une boite dans laquelle on peut ranger une seule valeur. Pour pouvoir retrouver et utiliser facilement le contenu de cette boite, on va lui adjoindre un nom : c'est le nom de la variable. Et comme tout est très rustique en informatique, il faudra préciser de quel type est la valeur que l'on a l'intention de mettre dans la boite: c'est le type de donnée.

La déclaration se fait dans un paragraphe qui commence par VAR. Par exemple déclarer une variable telle que RESULTAT : Integer; veut dire que l'on va utiliser une variable nommée RESULTAT qui ne pourra recevoir que des valeurs de type INTEGER (Entier).

Attention, Chaque fois qu'une nouvelle valeur est placée dans la variable, elle détruit la valeur qui s'y trouvait précédemment.

Les principaux types de données qui peuvent être utilisés pour déclarer des variables sont définis dans le tableau suivant:

Type	*Type PASCAL*	*nb d'octet*	*intervalle de définition*
ENTIER	BYTE	1	0 à 255
ENTIER	WORD	2	0 à 65535
ENTIER	INTEGER	2	-32768 à 32767
ENTIER	LONGINT	4	-2147483648 à 2147483647
REEL	REAL	6	$2.9\ 10^{-39}$ à $1.7\ 10^{+38}$
CARACTERE	CHAR	1	0 à 255 suivant code ASCII
CHAINE	STRING	256	255 carcatères maxi.

Lorsqu'on utilise une chaine de caractères, on peut atteindre séparément chacun des caractères en utilisant la syntaxe:

S[1] qui représente le premier caractère de la chaine S.
S[25] qui représente le 25ème caractère de la chaine S.

Constantes

Les constantes sont déclarées dans un paragraphe qui commence par CONST . A la différence des variables la valeur qui est placée dans la constante au moment de la déclaration ne pourra plus être modifiée au cours du déroulement du programme. Si l'on déclare ANNEE = 1995, on peut dire en simplifiant que chaque fois que le programme rencontre l'identificateur ANNEE, il le remplace par la valeur 1995. L'utilisation des constantes est particulièrement intéressante lorsque l'on veut paramétrer un programme.

Le Programme principal

Limité le plus souvent à quelques lignes, il constitue le point de départ du programme puisque le programme sera exécuté en commencant par la première instruction du programme principal.

Exemple

Ecrire un programme qui demande le nom et le prénom de l'utilisateur et affiche ses initiales. Frappez ce programme à l'aide de l'éditeur de Turbo-Pascal, compilez le et faites des essais pour voir la réaction du compilateur ainsi que les différents résultats obtenus.

```
Program Initiales;
Uses CRT;
```

```
Var
      Nom,Prenom : String;

Begin
      Writeln (‘Donnez votre nom : ‘);
      Readln (Nom);
      Writeln (‘Donnez votre prénom : ‘);
      Readln (Prenom);
      Writeln (‘Vos initiales sont : ‘,Nom[1],’.’,Prenom[1],’.’);
end.
```

Quelques explications : On rencontre deux instructions dans ce programme Writeln et Readln.

Writeln : Cette instruction (il faut dire procédure) permet d'écrire à l'écran. On peut écrire soit un texte fixe (Constante), soit le contenu d'une variable.
Exemple : Writeln ('Bonjour'); écrit le texte qui est contenu entre les guillemets. Il s'agit d'une chaine de caractères constante. Les constantes chaine de caractères sont toujours encadrées d'apostrophes. Writeln (Nom); écrit à l'écran le contenu de la variable Nom. Il n'y a pas d'apostrophe. On peut combiner les deux mode d'écriture en séparant les différents affichage par des virgules comme ceci : Writeln ('Bonjour ',Nom); qui écrira à la suite la constante bonjour et le contenu de la variable Nom.

Readln : cette instruction permet de lire une donnée au clavier. L'instruction bloque le déroulement du programme et stocke les caractères tapés au clavier dans la variable désignée entre parenthèses. La saisie des caractères s'arrête lorsqu'on appuie sur la touche ENTREE et le programme continue alors de se dérouler normalement. Readln permet de saisir aussi bien des chaines de caractères que des nombres de tout type.

Voilà quelques notions de base suffisantes pour écrire nos premiers programmes. Essayons un exercice un peu plus difficile qui se répétera assez souvent dans les chapitres qui suivent. Voici un algorithme, rédigé en Français, que vous devez traduire en Turbo-Pascal et tester sur la machine.Il vous appartient de définir le type des variables utilisées.

Traduction

Ecrire un programme qui calcule la moyenne de deux nombres réels.

```
Algo
      Demander le premier nombre (variable Nombre1)
      Demander le second nombre (variable Nombre2)
      Afficher le résultat du calcul de la moyenne.
FinAlgo.
```

L'affectation

On vient de le voir, on peut saisir au clavier une valeur qui sera placée dans la variable désignée. Mais il faut aussi pouvoir placer une valeur dans une variable sans passer nécessairement par le clavier. Ce sera le rôle du signe d'affectation.

Nombre1 **:=** 25;
Prenom **:=** 'Pierre';

Le signe **:=** peut se lire "reçoit la valeur". Ainsi, les deux lignes précédentes peuvent se lire la variable Nombre1 reçoit la valeur 25 ou la variable Prenom := la veleur Pierre.

Exemples

Le programme qui suit ne demande aucune intervention. Il fonctionne tout seul! Vous remarquerez lors de la première exécution que la valeur N1 contient une valeur quelconque. C'est toujours le cas en Pascal. Une variable qui n'a jamais été affectée contient une valeur quelconque.

```
Program Seul;
Uses Crt;

Var
      N1, N2 : Real;

begin
      Writeln  ('N1 contient : ',N1);
      N1 := 25.56;
      Writeln  ('N1 contient : ',N1);
      N2 := 10;
      Writeln ('Résultat : ',N1*N2:10:2);
end.
```

On peut aussi, naturellement, utiliser les deux moyens d'affectation (Readln et :=) dans un même programme.

```
Program LesDeux;
Uses Crt;

Var
      Nom : String;
      Taille,poids,Difference : word;
begin
      Difference := 105;
      Writeln ('Votre nom : ');
      Readln (Nom);
      Writeln ('Quelle est votre taille (en cm) : ');
```

```
        Readln (taille);
        Writeln ('Votre poids devrait être de ', taille - Difference);
end.
```

Exercices :

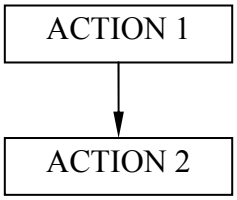
Faire les exercices du chapitre STRUCTURES de PROGRAMME.

LES STRUCTURES DE CONTROLE

Les structures de contrôle permettent au programmeur de réaliser l'enchaînement des instructions, de prendre en compte des choix ou de faire répéter une suite d'instructions. Ces structures, communes à la plupart des langages, sont le fondement de la programmation et doivent être impérativement maitrisées.

L'enchaînement

Tous les langages possèdent une instruction qui permet d'enchaîner deux instructions, c'est à dire de faire suivre une instruction par une autre et de permettre leur exécution séquentielle. Elle sera notée en Turbo-Pascal par le signe de ponctuation ";" .



La figure 2 illustre ce phénomène d'enchainement et montre qu'un programme n'est qu'une suite d'instructions enchainées.

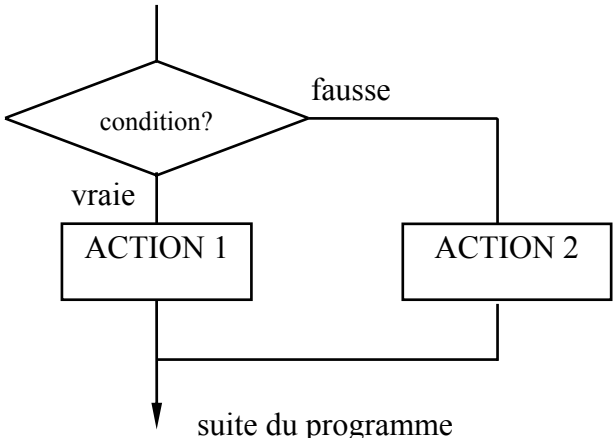
```
Var
        I,J,Ad  : Integer;        {déclaration de trois variables de type entier}

Begin
        I  := 3;                  {I recoit la valeur 3}
        J  := 4;          {J recoit la valeur 4}
        Ad := I+J         {Cette instruction n'est pas suivie du signe ; car}
End.                      {il n'y a pas d'enchainement sur une autre instruction}
```

Remarque : le signe := représente l'opérateur d'affectation. Il faut lire dans I:=3 que la variable I recoit la valeur 3. On peut, de la même manière, affecter une variable chaine de caractère. Exemple : Phrase := 'Le soleil brille'.

Structure alternative

Structure SI...ALORS...SINON

Cette structure permet de tenir compte de l'état d'une condition et de différencier les actions à réaliser suivant que la condition testée est vraie ou fausse. L'organigramme ci-dessous permet d'illustrer ce propos:



En Turbo-Pascal, Cette structure se traduira de la façon suivante:

IF *condition* **THEN** *action 1* **ELSE** *Action 2*

Exemple

Le programme demande deux nombres réels et en fait la division si c'est possible. Dans le cas contraire, le programme indique que la division est impossible.

```
Program Division;
Uses Crt;
Var
        Nombre1, Nombre2 : Real;

Begin
        Writeln ('Quel est le premier nombre? : ');
        Readln (Nombre1);
        Writeln ('Quel est le second nombre? : ');
        Readln (Nombre2);
        If Nombre2 = 0   then Writeln ('Division par 0 impossible!')
                         else Writeln ('Résultat = ',Nombre1 / Nombre2:10:2);
End.
```

ATTENTION. Il n'y a pas de point-virgule à la fin de la ligne du If. C'est normal puisque l'instruction ne se termine pas là. Elle ne se termine qu'à la fin de la ligne suivante...après le Else.

Notez également que l'indication placée après la division à savoir :10:2 permet de formater l'affichage du nombre obtenu en résultat. Ce nombre sera, dans le cas présent, affiché sur une zone de 10 caractères et avec 2 décimales. Pour un réel, s'il n'y a pas de format d'affichage de précisé, le nombre sera affiché en notation scientifique.
Dans le cas d'un nombre entier, il suffit d'indiquer le longueur de la zone d'affichage comme par exemple Nombre:8 qui affichera le nombre sur 8 caractères en alignant les unités sur le caractère le plus à droite.

Structure réduite

On peut naturellement disposer de la structure réduite Si..Alors qui permet de traiter un seul cas de l'alternative. Le Sinon étant implicitement traité par enchainement de l'instruction suivante dans le programme.

```
Var
      C      : Char;

Begin
      Write ('Tapez une lettre : ');
      Readln (C);
      If (C < 'A') or (C > 'Z') then Writeln ('Ce n''est pas une lettre majuscule');
      Write (C);
End.
```

Condition multiple

On peut, comme dans le programme précédent, avoir à indiquer plusieurs conditions à satisfaire simultanémént. Chaque condition devra être placée entre parenthèses et les différentes conditions seront associées grâce à des opérateurs logiques. Le plus souvent, il s'agira du AND (ET logique) et du OR (OU logique).

```
If (a>b) and (a<d) then ....
If ((a>b) and (a<c)) or ((a>d) and (a<=e)) then ....
```

Le signe <= signifie inférieur ou égal comme le signe >= signifie supérieur ou égal.

Exercices

Faire les exercices du chapitre : STRUCTURE ALTERNATIVE.

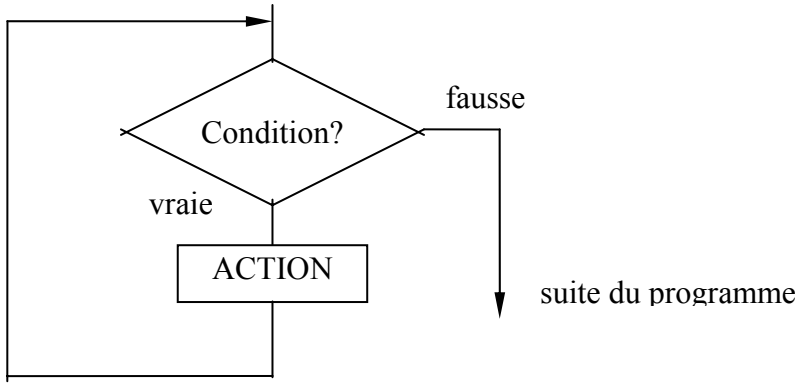
Structure de répétition

Structure TANTQUE...FAIRE...

L'ordinateur est apte à répéter un grand nombre de fois un même traitement. Il peut d'ailleurs le répéter à l'infini si on ne lui indique pas correctement la façon de cesser de le faire. La structure répétitive se présente sous la forme suivante:

While *condition* do *Action*

La condition est une expression logique susceptible de prendre la valeur Vrai ou Faux. Il est évident que l'action doit modifier l'un des éléments intervenant dans la condition, sans quoi la condition n'est jamais modifiée et l'action se répéte à l'infini. L'organigramme suivant illustre clairement ce phénomène.



Supposons que l'on veuille lire les noms et prénoms des individus contenus dans un fichier, on pourra écrire le programme suivant:

```
Var
        Nom,Prenom  : String;
        N,N_Fiche     : Integer;

Begin
        N:=1;                           {Initialiser le compteur de fiche à 1}
        Write ('Nombre de fiches à lire');
        Readln (N_Fiche);               {Combien de fiches à saisir ?}
        While N <= N_Fiche do
        begin
                Write ('Nom de la fiche',N,' :');Readln (nom);
                Write ('Prénom de la fiche',N,' : ');Readln (prenom);
                N := N + 1;             {Cette action modifie la condition}
        end;
End.
```

On peut remarquer que, dans la structure Tant que, l'action peut ne pas être exécutée si la condition se révèle fausse dés le premier test. Dans l'exemple précedent, s'il n'y a pas de fiche à lire (N_Fiche = 0), on n'éxécute jamais les actions de lire car la condition ((N <= N_Fiche) est fausse dés le premier test. Il sera souvent nécessaire, avant une boucle Tant que, d'initialiser les paramètres pour que la condition soit vraie lors du premier test, d'autant que le contenu des variables est tout à fait aléatoire si on ne les a pas encore utilisées.

```
Var
        Reponse : String;

Begin
        Réponse := 'AAA';                    {Initialisation de la variablede test}
        While (Reponse<>'OUI') and (Réponse<>'NON') do
        begin
                Write ('Donnez votre réponse (OUI/NON) : ');
                Readln   (Reponse);
        end;
End.
```

Traduction

Traduire l'algorithme suivant qui demande à l'utilisateur d'entrer au clavier un nombre entier compris entre 0 et 15 inclus.

```
Algo
    Demander un nombre (Nb)
    Tantque Nb n'est pas compris entre 0 et 15 inclus répéte
        Demander un nouveau nombre (Nb)
    Ftq   {fin du Tantque}
    Afficher le nombre (Nb)
FinAlgo
```

Exercices

Faire les exercices du chapitre : STRUCTURE REPETITIVE TANT QUE ...FAIRE

STRUCTURES COMPLEMENTAIRES

Les deux structures que nous venons de voir permettent d'écrire n'importe quel programme, aussi complexe soit-il. Mais il peut s'avèrer que dans certains cas, il soit plus commode d'utiliser des formes dérivées de ces structures de base.

Structure alternative

Structure Case...Of

Souvent, on veut choisir non plus entre deux possibilités comme avec la structure SI..Alors...Sinon, mais entre plusieurs. On utilisera alors la structure Case...of qui s'écrira ainsi.

```
Case <variable> of
        i1 : Action1;
        i2 : Action2;
        i3 : Action3;
        .........
        in : Actionn
        Else : Action0
end {fin de Case .. of}
```

Suivant l'état de la variable c'est l'action correspondante qui sera exécutée. Si la variable a la valeur i2 c'est l'action2 qui sera exécutée. Si aucun état ne correspond à l'état de la variable, alors c'est l'action Else qui est exécutée[1]. A partir du moment où une action a été exécutée, on sort de la structure et aucun autre état de la variable ne sera testé.

La variable doit nécessairement être du type scalaire ou intervalle.

```
Var
        C : Char;

Begin
        Write ('Tapez une lettre O,o,Y,y,N,n'); Readln (C);
        Case C of
                'O' : Writeln ('Réponse donnée : OUI');
                'o' : Writeln ('Réponse donnée : oui');
                'y' : Writeln ('Réponse donnée : yes');
```

[1] A noter que 'Autre cas' est facultatif dans l'écriture de la structure et qu'il n'existe pas dans tous les langages

```
                'Y' : Writeln ('Réponse donnée : YES');
                'n' : Writeln ('Réponse donnée : non');
                'N' : Writeln ('Réponse donnée : NON');
                Else : Writeln ('Réponse incorrecte')
        End
End.
```

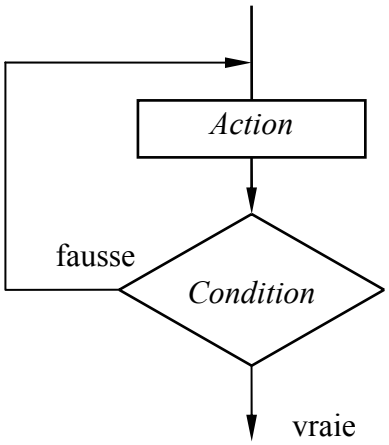
Structures répétitives

Structure REPEAT ... UNTIL...

Cette autre forme de répétition, trés voisine de While...Do, est la boucle :

Repeat *Action* Until *Condition*

L'organigramme qui suit montre le fonctionnement de ce type de boucle. La différence essentielle avec la boucle Tantque est que ***l'action sera exécutée au moins une fois*** ce qui n'impose pas d'initialiser à une valeur précise les variables de test. Mais le fait que l'action soit obligatoirement executée pourra, dans certains cas, être un inconvénient.




```
Var
        Reponse         : String;

Begin
        Repeat
                Write ('Donnez votre réponse (OUI/NON) : ');
                Readln   (Reponse);
        Until (Reponse = 'OUI') or (Reponse = 'NON')
End.
```

On peut remarquer que la condition de la boucle Tantque n'est rien d'autre que la condition de la boucle Répéter à laquelle on a appliqué le théorème de DE MORGAN.

La lecture controlée d'une chaine de caractères ne devant contenir que des minuscules et au plus 150 caractères peut s'écrire de la façon suivante:

```
Var
        ch              : String;
        C               : Char;
        Longueur        : Integer;

Begin
        Ch := '';               {on s'assure que la chaine de saisie est vide}

        Longueur := 0;          {Initialisation de la variable de comptage}
        Repeat
                Readln (C);  {Lire un caractère}
                If (C > 'a') and (C < 'z') Then
                begin
                        Ch := Ch + C;                   {accepter le caractère s'il est correct}
                        Longueur := Longueur + 1;       {mémoriser le nb de caractères}
                end;
        Until (Ord (C) = 13) or (Longueur = 150);
                        {si on tape la touche RETOUR  ou si Longueur=150}
        Writeln ('Caractères saisis : ',Ch);
End.
```

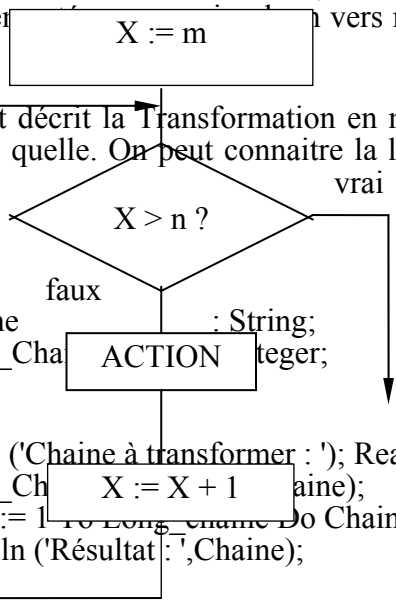
Structure FOR...TO...DO

Dans une boucle Tantque ou Répéter, on ne connait pas par avance le nombre de boucles qui devront être faites. Dans certains cas, pourtant, on connait avec exactitude le nombre d'itérations à réaliser. Cette forme de boucle nous permettra de le programmer.

For *x variant de m* To (ou DownTo) *n* Do

Il est important de noter que contrairement à la boucle Tantque, ce n'est pas au programmeur de traiter l'incrémentation de la variable de boucle x. D'ailleurs, dans la plupart des langages, il est formellement interdit de tenter de modifier la valeur de cette variable à l'intérieur de la boucle.

Si l'option DownTo est notée dans la boucle, x sera décrémentée pour varier de m vers n, sinon x sera incrémentée pour varier de m vers n. Par défaut x sera sytématiquement incrémentée.



L'algorithme suivant décrit la Transformation en majuscule d'une chaine de caractères qui a été saisie telle quelle. On peut connaitre la longueur de la chaine par la fonction Length (Chaine).

```
Var
    Chaine          : String;
    Long_Chaine     : Integer;
begin
    Write ('Chaine à transformer : '); Readln (Chaine);
    Long_Chaine := Length (Chaine);
    For I := 1 To Long_Chaine Do Chaine[i] := Upcase (Chaine[i]);
    Writeln ('Résultat : ',Chaine);
End.
```

Exercices

Faire les exercices du chapitre : STRUCTURES COMPLEMENTAIRES.

FONCTIONS ET PROCEDURES

Les procédures et les fonctions, on peut dire aussi les sous-programmes, fournissent au programmeur un moyen commode d'abstraction en lui permettant de nommer une séquence complexe d'instructions et de l'appeler autant de fois qu'il sera nécessaire au cours d'un même programme. En outre, les langages tels que Pascal permettent de 'passer' des données à la procédure ou à la fonction. La valeur de ces données pouvant varier d'un appel à l'autre on appelera tout naturellement ce mécanisme un "passage de paramètre". Il est très important de comprendre ce mécanisme et nous y reviendrons largement dans les pages qui suivent.

Au départ, les procédures furent inventées pour enrichir d'instructions nouvelles, le langage de base fourni par l'éditeur, mais elles permettent aussi d'écrire des programmes structurés qui sont donc trés lisibles. Si la structuration facilite la lecture des programmes, elle facilite aussi leur maintenance et cet aspect est particulièrement important si l'on se place dans la perspective d'un développement professionnel de logiciel.

Pouvoir enrichir un langage, c'est bien, mais à condition de pouvoir facilement réutiliser ces nouvelles instructions. C'est le rôle des bibliothèques (ou unités précompilées). On pourra en effet, rassembler dans des bibliothèques tout un ensemble d'instructions, et les mettre à disposition de tous les programmes. Turbo-Pascal saura également gérer automatiquement les mises à jours de ces bibliothèques pour les répercuter sur les programmes qui les utilise. Nous créerons assez rapidement une bibliothèque. Il serait dommage de se priver d'une telle possibilité.

On peut considérer qu'une procédure ou une fonction est un programme et de ce fait on retrouvera dans leur structure, celle d'un programme. Les seules différences importantes sont les notions de paramètres et de variables locales.

Nous verrons dans un premier chapitre les fonctions, puis dans un second les procédures. Ces deux chapitres présentes des difficultés réelles de compréhension et il ne faudra pas hésiter à relire plusieurs fois chaque paragraphe.

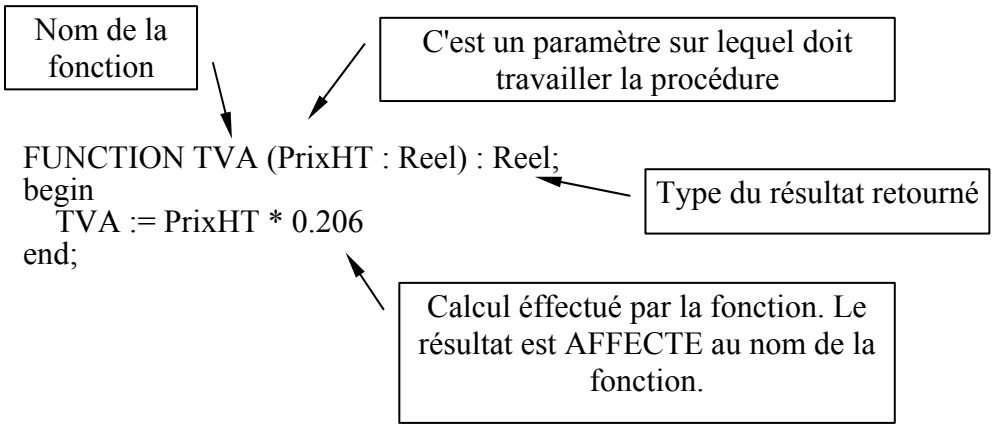
LES FONCTIONS

Définition

Une fonction est un ensemble d'instructions qui forment un sous programme. Une fonction en programmation fonctionne un peu comme en mathématiques. Chaque fois qu'on l'appelle elle renvoie (ou elle retourne) au programme appelant une valeur qui est le résultat du traitement effectué par les instructions de la fonction. Sin(x), Exp(x) sont des fonctions. Les fonctions en programmation ne se limitent pas à des calculs numériques. Un fonction peut renvoyer tout type de base, c'est à dire un nombre, un caractère ou une chaines de caractères.

Déclaration

La déclaration d'une fonction se fait ainsi (c'est ici le cas le plus simple):



Une fonction se déclare en commencant par le mot réservé FUNCTION suivi du nom de la fonction qui est choisi par le programmeur. On trouve ensuite éventuellement un ou plusieurs paramètres qui sont notés entre parenthèses. Ces paramètres servent à transmettre à la fonction les valeurs sur lesquelles les instructions de la fonction doivent travailler. On trouve ensuite un signe deux points (:) derrière lequel on note le type du résultat qui sera renvoyé par la fonction.

On déclarera ensuite entre Begin et End, la liste des instructions nécessaires pour effectuer le traitement désiré.

IMPORTANT: Il ne faudra jamais oublier en fin de fonction d'affecter le résultat du traitement au nom de la fonction. Sans quoi, votre fonction ne renverra aucun résultat.

Utilisation

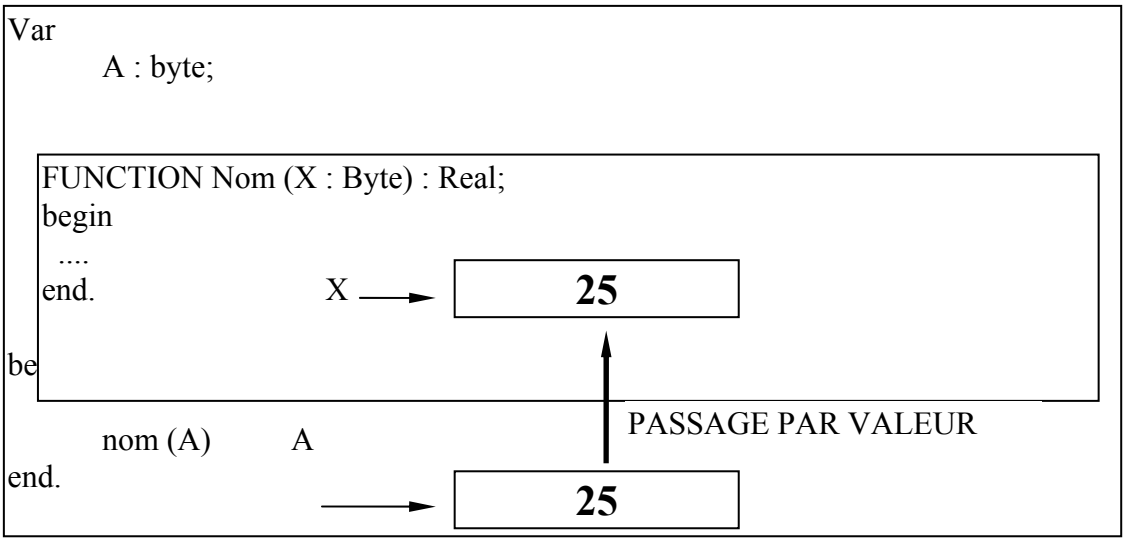
Une fonction, une fois déclarée, pourra être appelée soit depuis le programme principal, soit depuis une autre fonction. Le programme complet se présentera de la façon suivante, sachant que l'on déclare les fonctions juste après les variables.

```
Program Calcul_TVA;                    {En-tête du programme}
Uses CRT;

Var                                    {déclaration des variables}
        Prix,PrixTTC : Real;

FUNCTION TVA (PrixHT : Reel) : Reel;   {déclaration et description de la fonction}
begin
        TVA := PrixHT * 0.206
end;

begin                                  {programme principal}
        Writeln ('Quel le montant du prix hors taxe? : ');
        Readln (Prix);
        PrixTTC := Prix + TVA(Prix);
        Writeln (Prix toutes taxes : ',PrixTTC:10:2, ' dont TVA : ',TVA (Prix):10:2);
end.
```

Il faut remarquer que la fonction s'utilise comme on utiliserait une variable du type de la valeur retournée par la fonction. On peut l'utiliser dans un calcul, si la fonction retourne une valeur numérique ou dans un affichage etc...

Passage de paramètre par valeur

L'exemple qui précède montre que l'on peut passer des valeurs à une fonction. Le mécanisme de transmission doit être bien compris et le schéma qui suit tente de l'illustrer .



Le plus grand cadre représente le programme principal avec la déclaration de la variable globale A. Le cadre intérieur représente la fonction avec la déclaration de son paramètre X.
Au moment de l'appel il se passe deux choses:

1. Création d'une variable locale (voir paragraphe suivant) nommée X.
2. Recopie de la valeur du paramètre dans cette nouvelle variable.

A la fin de la fonction la variable X est détruite et la valeur qui s'y trouvait est perdue. On peut donc modifier en toute tranquilité le contenu de la variable X sans affecter la valeur originale qui se trouve dans A. En quelque sorte on travaille sur une copie du paramètre.

Variable locale

Jusqu'à présent nous avons déclaré des variables dans le paragraphe VAR qui se trouve en tête de programme. Et nous avons en fait déclaré des **variables globales**. Ces variables, dites globales, sont disponibles et peuvent donc être utilisées dans tout le programme y compris dans les fonctions.

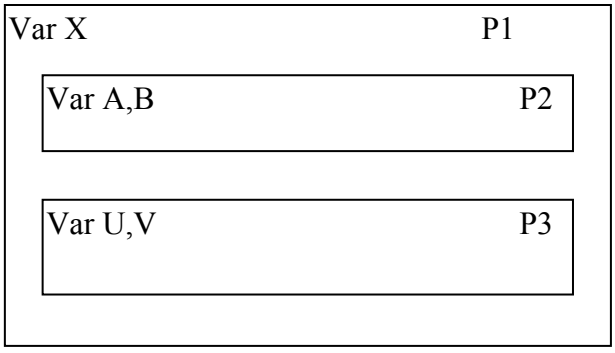
Souvent, on aura besoin d'une variable uniquement à l'intérieur d'un fonction. Il faudra alors déclarer des **variables locales**. Ces variables ne seront utilisables qu'à l'intérieur de la fonction et seront totalement inconnues à l'extérieur de la fonction. Ce statut permettra en particulier d'utiliser plusieurs fois le même nom de variable dans des fonctions différentes. Une variable locale se déclarera tout simplement dans un paragraphe VAR à l'intérieur de la fonction.

Si P1 représente le programme principal et P2, P3 deux fonctions, le schéma qui suit illustre le propos précédent.

Dans P1 seule X est connue

Dans P2 on dispose de A,B et X

Dans P3 on dispose de U,V et X



Exemple

On désire disposer d'une fonction qui calcule le troisième terme d'une suite arithmétique dont on connait le premier terme et la raison.

```
Program Suite;
Uses CRT;
```

```
Var
        Raison,Premier : Integer;       {déclaration des variables globales}

FUNCTION Terme3 (P,R : Integer) : Integer;
Var
        I,Resultat : Integer;           {déclaration des variables locales}
begin
        Resultat := P;
        For i:=1 to 2 do Resultat := Resultat + R;
        Terme3 := Resultat;             {affectation du résultat au nom de la fonction}
end;

begin
        Writeln ('premier terme : ');Readln (Premier);
        Writeln (('raison : '); Readln (Raison);
        Writeln (le troisième terme de la suite est : ',Terme3 (Premier,raison));
end.
```

Traduction

On désire écrire une fonction qui recoit deux paramètres: Une chaine de caractères et un caractère. Après traitement elle retourne un nombre qui représente le nombre de fois que le caractère se trouve dans la chaine de caractère.

- Définir l'en-tête de la fonction
- Ecrire les instructions nécessaires en respectant l'algorithme suivant:

Mettre un compteur à 0
Pour chaque caractère de la chaine
Si le caractère de la chaine est égal au paramètre
alors Incrémenter le compteur.

- Affecter le résultat au nom de la fonction
- Ecrire un petit programme principal de test.

Exercices

Faire les exercices du chapitre : LES FONCTIONS.

LES PROCEDURES

GENERALITES

Les procédures jouent un rôle assez proche de celui des fonctions, mais en utilisant des moyens différents pour restituer les résultats du traitement réalisé par la procédure. En effet une procédure, contrairement à une fonction, n'est pas obligée de renvoyer un résultat. Elle peut simplement effectuer un traitement comme le montre l'exemple qui suit. Mais elle peut tout aussi bien renvoyer plusieurs résultats. C'est le mode de passage des paramètres qui va permettre d'utiliser ces diverses possibilités.

Les procédures sans paramètres

Déclaration et appel

Le moyen le plus simple de définir une procédure est de la définir uniquement par son nom. On dira alors que c'est une procédure sans paramètre.Il suffira pour l'appeler d'évoquer ce même nom.

```
Procedure Affiche_3Oui;      {En-tete de procédure}
Var                          {Déclaration des variables locales}
   I    : Integer;
Begin                        {Corps de la procédure}
   For I :=1 to 3 do Write ('oui ');
end;
```

La procédure qui vient d'être définie ne fait rien d'autre que d'afficher 3 fois le mot Oui. On appelera cette procédure en incluant, dans le programme principal, ou dans une autre procédure, la ligne suivante :

```
   Affiche_3Oui;
```

ce qui aura pour effet d'interrompre le déroulement du programme principal pour exécuter les instructions de la procédure. Lorsque celles-ci sont exécutées, on revient automatiquement à la poursuite du programme appelant (programme principal ou procédure).

La forme complète du programme sera:

```
Program 3_Oui;
Uses Crt;

Procedure Affiche_3Oui;
Var
      I       : Integer;
Begin
```

```
        For I :=1 to 3 do Write ('oui ');
end;

begin
  ClrScr;
  Affiche_3Oui;
end.
```

Traduction

On souhaite réaliser une procédure qui encadre l'écran avec un cadre composé de caractères semi graphiques (codes ASCII de 129 à 256). On suivra l'algorithme suivant, sachant qu'un écran comporte 25 lignes de 80 caractères:

Positionner curseur (voir Gotoxy)
Afficher le coin en haut et à gauche
Pour i variant de 1 à 79
    afficher un tiret horizontal
Afficher le coin en haut et à droite
Pour i variant de 2 à 23
    positionner le curseur
    afficher un trait vertical dans la première colonne sur la ligne i
    Positionner le curseur
    afficher un trait vertical dans la dernière colonne sur la ligne i
Positionner le curseur
afficher le coin en bas à gauche
Pour i variant de 1 à 78
    afficher un tiret horizontal
afficher le coin en bas à droite.

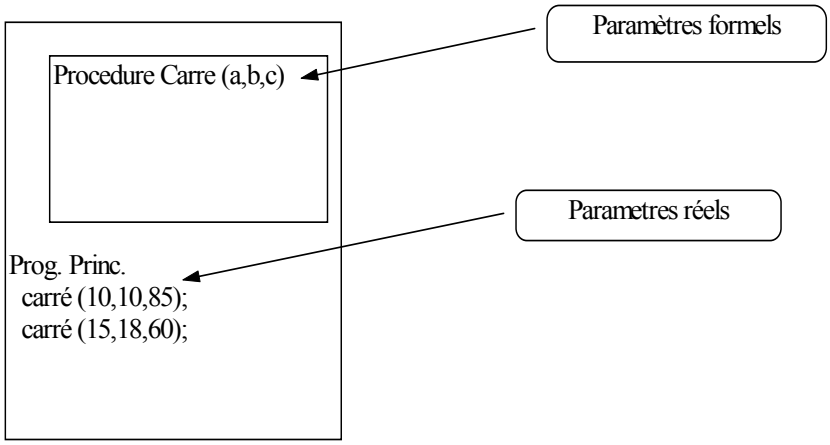
Exercices

Faire les exercices du chapitre : LES PROCEDURES SANS PARAMETRE.

Les procédures avec paramètres

Lorsqu'un programme appelle une procédure, des objets peuvent être transmis entre l'un et l'autre. Certains sont des données dont la valeur est transmise à la procédure. D'autres sont des résultats transmis de la procédure vers le programme appelant.

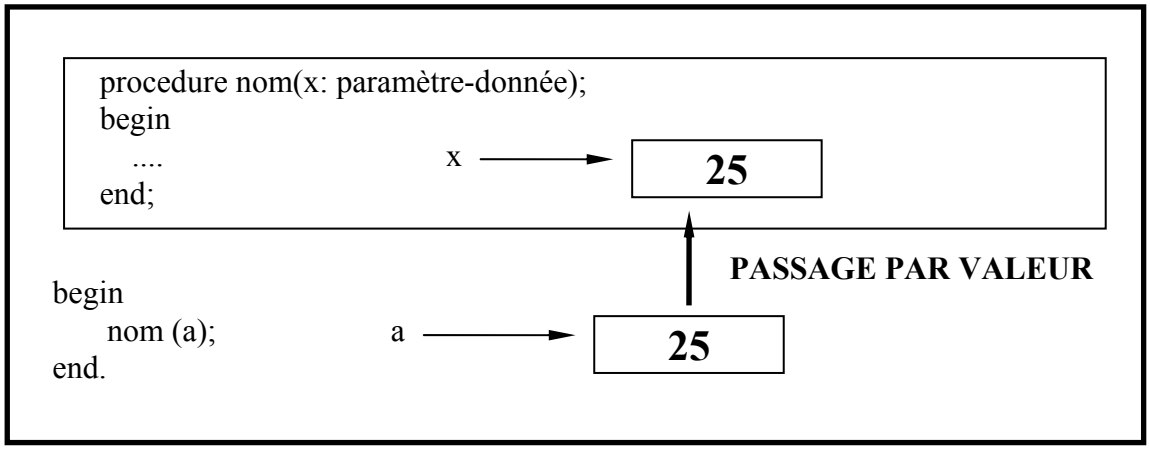
Les objets transmis vers la procédure sont dits **paramètres réels** ou arguments réels. A chacun d'eux doit correspondre un **paramètre formel** ou argument formel dans la procédure. Le schéma qui suit représente un programme principal (le grand rectangle et une procédure (petit rectangle) qui est appelée deux fois à parti du programme principal. Lors du premier appel, les paramètres formels a, b et c vont recevoir respectivement les valeurs 10, 10 et 85. Lors du deuxième appel les paramètres formels recevront alors les valeurs 15, 18 et 60. Il existe deux moyens de transmettre des paramètres: Le passage par valeur et le passage par référence. La différence entre les deux méthodes est très importante à saisir les deux paragraphes qui suivent expliquent cette différence.



Passage par valeur

Dans ce premier cas, lorsque que l'on dit que les paramètres formels recoivent une valeur, il faut comprendre que a, b et c sont des variables locales à la procédure qui sont automatiquement initialisées aux valeurs des paramètres réels. Les paramètres formels pourront être utilisés à l'intérieur de la procédure comme des variables locales normales. N'oublions pas que ces variables sont détruites à la fin de la procédure, donc les valeurs qu'elles contiennent sont perdues.

On peut dire, d'une autre manière, qu'un paramètre est transmis par valeur (paramètre donnée) lorsque la procédure utilise une copie de la donnée pour travailler. De ce fait toute modification du paramètre formel à l'intérieur de la procédure ne modifie en aucune manière le paramètre réel (valeur de la variable dans le programme appelant). Le schéma qui suit montre en fait que l'on connait bien ce type de passage puisque c'est le même que pour la transmission des paramètres à une fonction.



A l'appel de son nom, avec la donnée a, la procédure crée une variable x ayant pour valeur celle de a. Les modifications éventuelles sur x n'interviennent pas sur a.

La procédure (incluse dans le programme Soulignement qui suit) permet de souligner une chaine de caractères. Les paramètres sont passés par valeur.

```
Program Soulignement;
Uses Crt;
Var
        Roi : String;

Procedure Souligner (t:string);
Var
        I   : Integer;
begin
        Writeln (t);
        For I:=1 to length (t) do
        Write ('-');
        Writeln
end;

begin
        Souligner ('Histoire');
        Roi := 'Louis XIV';
        Souligner (roi);
        Souligner ('Histoire de '+Roi);
end.
```

Les paramètres réels successifs sont : 'Histoire', 'Louis XIV' et 'Histoire de Louis XIV'.

Le paramètre formel est : t

Exercices

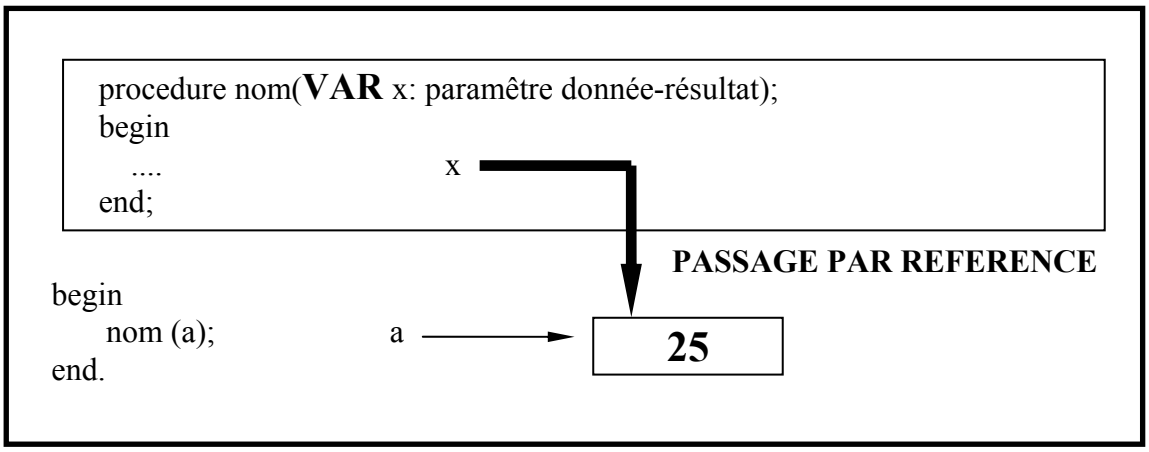
Faire les exercices du chapitre :
PROCEDURES AVEC PARAMETRES (passage par valeur)

Passage par référence

Dans un passage par référence, la procédure travaille directement sur le paramètre réel[1]. Aucune copie du paramètre réel n'est effectuée et toute modification du paramètre formel par la procédure entraine la modification du paramètre réel. Cette solution permet à la procédure de renvoyer un résultat au programme appelant. Ce qui explique que ce mode de passage porte aussi le nom de 'passage par paramètre-résultat'.

A l'appel de son nom, la procédure ne crée pas de variable locale, mais se contente de créer un nom de variable pointant sur le même contenu que a. Ce qui fait que chaque fois que l'on travaille sur la variable x, on travaille en fait sur la variable a et donc toute modification apportée à la variable x pendant le traitement de la procédure affecte la variable a. On indique ce mode de passage en ajoutant le mot Var devant le paramètre formel considéré. Les paramètres réels sont alors obligatoirement des variables.

1 C'est en fait l'adresse (la référence, ou le pointeur) de la variable qui est transmis à la procédure.



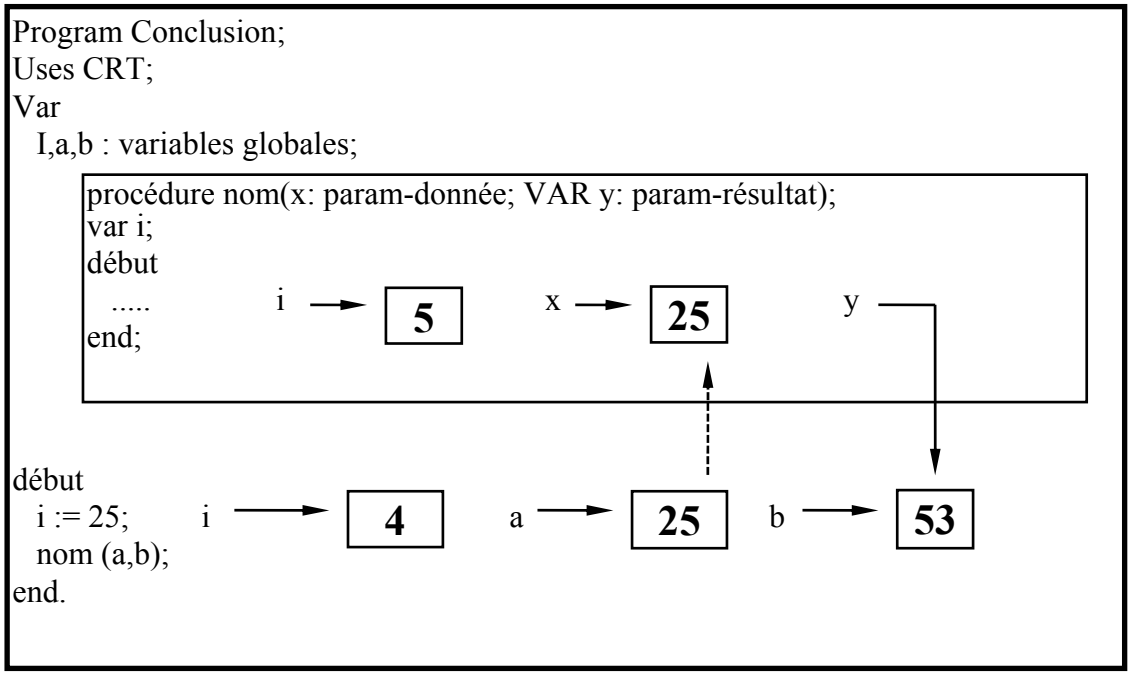
Exemple

Dans l'exemple qui suit la procédure Saisie utilise deux paramètres passés par référence. L'un permet de récupérer une chaine de caractère saisie au clavier et l'autre de connaitre le nombre de caractères qui ont été saisis

```
Uses CRT;
Var
        Chaine : String;
        Nb_Car : Byte;

Procedure Saisie (Var Ch : String; Var N : Byte);
Var
        C : Char;
Begin
        C:=Readkey;
        Ch := '';
        N := 0;
        While (C<>#13) do
        begin
                Write (C);
                Ch := Ch + C;
                Inc (N);
                C := Readkey;
        end;
end;

begin
        Writeln ('Donner une phrase : ');
        Saisie (Chaine,Nb_Car);
        Writeln (' La chaine saisie : ',Chaine,' contient ',Nb_Car,' caractères');
end.
```

En Conclusion

Toutes les notions vues dans les paragraphes précédents peuvent être utilisées simultanément. La figure qui suit permet d'en donner une illustration.



x, i, variables locales, n'ont d'existence que lors de l'appel de la procédure et n'ont aucune influence sur les variables du programme a, i, b.

a, variable globale est la même, dans la procédure et dans le programme principal

y, paramêtre-résultat, est crée lors de l'appel de la procédure mais utilise la même case mémoire que b; toute modification sur y entraine la même modification sur b.

Exercices

Faire les exercices du chapitre : LES PROCEDURES AVEC PARAMETRES (Passage par référence).

LA RECURSIVITE

On dit qu'une procédure est récursive lorsqu'apparaît à l'intérieur de sa définition un appel à cette même procédure. En d'autre termes, la procédure s'appelle elle même. Cette notion de récursivité est parfois difficile à saisir. Certains pensent 'récursifs' d'autres pas. La notion de récursivité est trés importante en informatique.

Calcul du factorielle

Un exemple de traitement récursif est le calcul de factoriels.

n! = n * (n-1) * (n-2) * 1

que l'on peut écrire

si n > 0 n! = n * (n-1)!

On sait d'autre part que

0! = 1

La fonction s'écrira donc trés simplement.

```
Fonction fact (n : entier) : entier;
debut
  Si n = 0  alors  Fact := 1
          sinon  Fact := n * fact (n-1)
fin;
```

On remarque qu'il existe une 'condition d'arrêt'. C'est l'existence de cette condition qui va bloquer le processus de récursion à la condition qu'elle soit atteinte de façon certaine. dans l'écriture de toute procédure récursive, il sera nécessaire de mettre en évidence d'une part la forme de l'appel récursif et d'autre part la condition d'arrêt.

La fonction de FIBONACCI

La fonction de FIBONACCI qui représente la croissance d'une population de lapins est définie de la façon suivante :

ø(0) = 1
ø(1) = 1
ø(n) = ø(n-1) + ø(n-2)

On reconnait que la fonction est récursive car elle apparaît à gauche et à droite du signe d'égalité.

```
Fonction Fibo (n : entier) : entier;
début
  Si n = 0 alors Fibo := 1
          sinon si n = 1
                alors Fibo := 1
                sinon Fibo := Fibo (n-1) + Fibo (n-2)
fin;
```

Exercices

Faire les exercices du chapitre : LE RECURSIVITE

LES TABLEAUX

Structures de données

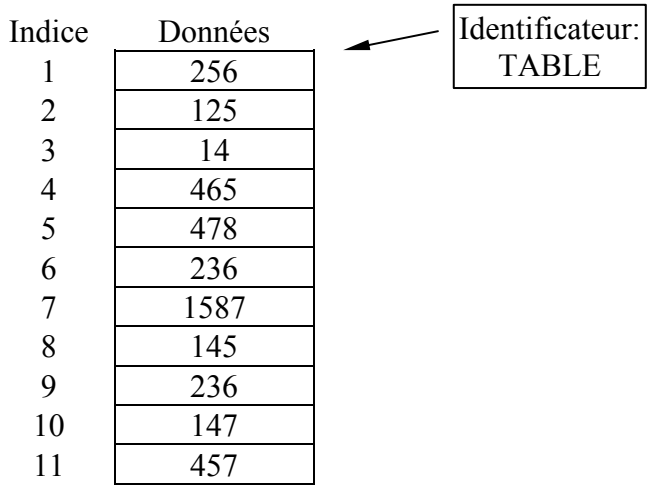
Si les structures de contrôle sont une composante essentielle de la programmation, l'organisation des données en une autre tout aussi importante. Organiser les données pour qu'elles soient facilement accessibles et utilisables demande de bien maitriser les structures de données. Nous avons utilisé jusqu'à présent des types de données fondamentales telles que Integer, Real ou Char. Ce sont des structures élémentaires qui ne permettent de manipuler qu'une seule valeur à l'aide d'un identificateur. Cette situation limite très rapidement les possibilités d'utiliser de grandes quantités de données.

Les programmeurs ont donc rapidement cherché à mettre en place des structures qui permettent de manipuler aisément des quantités importantes de données. Ils n'ont pas mis au point une mais plusieurs structures plus ou moins difficiles à maitriser et qui ont toutes leurs avantages et leurs inconvénients. Il n'existe pas de structure universelle. Il conviendra de choisir la structure la mieux adaptée à un problème donné.

Les principales structures que l'on peut rencontrer sont: Les tableaux que nous étudirons, mais aussi les piles, les files, les listes, les arbres et les graphes qui ne font pas l'objet d'un développement dans ce document car elles font le plus souvent à la notion de pointeur qui ne sera pas abordée non plus.

Qu'est ce qu'un tableau?

Un tableau est donc une structure de données. C'est à dire un ensemble qui comprend un identificateur (autrement dit un nom) et plusieurs données d'un même type. On accédera à chacune des données en utilisant un indice comme l'illustre le schéma qui suit et qui montre qu'à chaque indice correspond une donnée. La donnée d'indice 5 vaut 478. Cette façon de représenter un tableau est une convention, le rangement en mémoire des données et leur repérage par un indice se faisant très différemment. Dans cet exemple le tableau sera nommé TABLE. c'est l'identificateur qui est choisi par le programmeur selon les règles habituelles.

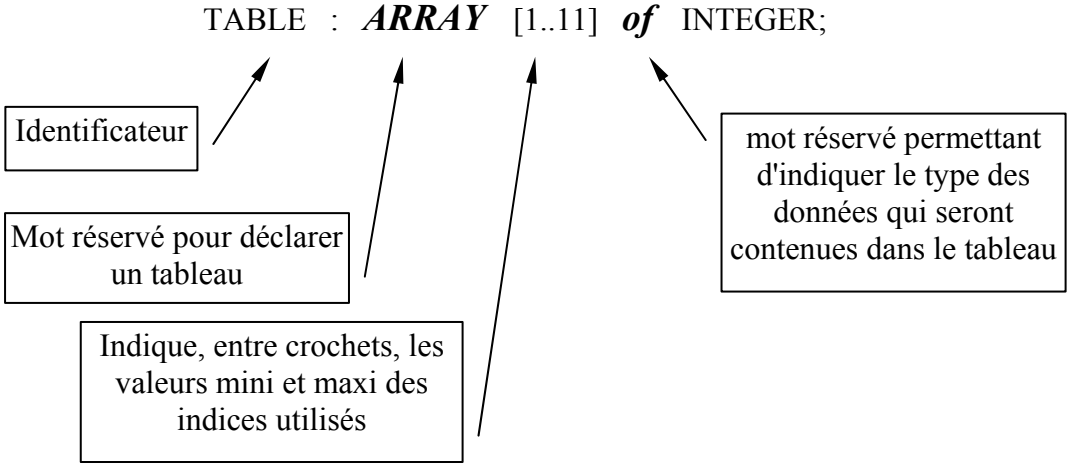


.

Déclaration d'un tableau

Syntaxe

Avant de pouvoir utiliser un tableau, il convient, comme pour une variable, de le déclarer. Cette déclaration se fait dans le paragraphe VAR et observe la syntaxe suivante:



Limites

Il existe deux limites à la déclaration d'un tableau.

La première, c'est le **nombre d'élèments**. Il est limité à 65535. On pourra donc au plus déclarer un tableau allant de [-32768..32767] ou [0..65735]. Il faut noter également que si le premier indice doit être inférieur au second, il peut tout à fait être différent de 0. On peut donc sans inconvénient déclarer les indices [1950..1999] qui définit un tableau de 50 éléments.

La seconde, et dans l'état actuel de nos connaissances, est la **place occupée en mémoire**. Le tableau ne doit pas dépasser 64 Ko. Il est facile de connaitre cette place en multipliant le nombre d'élèments par la taille d'un élèment. par exemple:

```
Table1 : Array [1..100] of Word;      occupe 100 * 2octets soit 200 octets
Liste : Array [1..100] of String;     occupe 100 * 255 octets soit 25500 octets
```

Utilistaion d'un tableau

Les éléments d'un tableau sont des variables et peuvent donc s'utiliser de la même manière. Seule la manière d'écrire diffère. En effet, il faudra toujours indquer à la suite de l'identificateur l'indice de la valeur utilisée. Exemples si l'on utilise le tableau Table1 déclaré ci-dessus:

```
Affectation d'une valeur                Table1 [15] := 1223;
Ecriture d'une valeur à l'écran         Writeln (Table1[18]);
Acquisition d'une valeur au clavier     Readln (table1[20]);
Affectation interne                     Table1[10] := Table1[15]);
Affectation globale d'un tableau        Table2 := Table1;
```

Dans ce dernier cas, il faut que les deux tableaux soient déclarés de la même manière. Toutes les valeurs de table1 seront affectées aux indices correspondant de Table2.

Passage d'un tableau en paramètre

Il est possible de passer un tableau comme paramètre à une procédure ou à une fonction. Ce passage peut se faire soit par valeur soit par adresse, mais nécessite une déclaration préalable de Type. Il est en effet impossible d'utiliser l'écriture suivante:

```
Procedure Essai_Tableau (Var T : Array [1..50] of Char);
```

Pour utiliser un tableau comme paramètre il faudra au préalable, déclarer un type. Ceci se fait dans un paragraphe spécial qui débute par le mot réservé TYPE et qui se place, en général, avant le paragraphe VAR.

```
TYPE
    T_Toto = Array[1..20] of string;

VAR
    Table : T_Toto;
```

```
Procedure Essai_Tableau (Var T : T_Toto);
begin
....
end;
```

Le type déclaré va permettre, comme on peut le voir ci-dessus, de déclarer d'une part des varaibles utilisées dans le programme soit en variables locales, soit en variables globales et d'autre part de déclarer les paramètres.

Le T_ qui précède le nom du type dans cet exemple n'est pas obligatoire mais peut permettre de répèrer rapidement dans un listing les types utilisés.

Exemple

```
Program Init_Tableau;
{ Ce programme initialise deux tableaux de 20 valeurs entières à des valeurs aléatoires
      entre 0 et 100 et affiche la somme des valeurs indice par indice}
Uses Crt;

TYPE
      T_Liste;

VAR
      Premiere_Liste, Deuxieme_Liste : T_Liste;

Procedure Init (Var T : T_Liste);
Var
      I : Byte;
Begin
      For I:=1 to 20 do T[i] := Random(100);
end;

Procedure Affiche (A,B : T_Liste);
Var
      I : Byte;
begin
      ClrScr;
      For I := 1 to 20 do Writeln ('Somme indice ',I:2,' : ',A[i]+B[i]:10);
end;

begin
      Randomize;
      Init (Premiere_Liste);
      Init (Deuxieme_Liste);
      Affiche (Premiere_Liste,Deuxieme_Liste);
end.
```

Tableaux à plusieurs dimensions

La structure de donnée que nous venons de voir est souvent dénommée Liste, vecteur ou Tableau à une dimension ce qui sous entend qu'il peut exister des tableaux à plusieurs dimensions capables de représenter des matrices par exemple. C'est en effet le cas, et l'on peut utiliser des tableaux jusqu'à 64 dimensions avec cette version de TurboPascal! Nous n'étudierons que les tableaux à deux dimensions, mais il sera facile de généraliser.

Représentation

Un tableau à deux dimensions est souvent représenté par une matrice. On trouvera donc deux indices: l'un représente les lignes de la matrice et l'autre les colonnes de cette même matrice. Il ne faut pas confondre cette représentation et la façon dont les données sont stockées en mémoire vive. Cela peut être très différent.

	1	2	3	4	5
1	1478				
2		1548			
3					
4			256		
5	1395				1478
6					
7				7895	

Déclaration et utilisation

On utilisera pour la déclaration, comme précédemment, le mot réservé Array.. of en indiquant entre crochets, la valeur des indices utilisés pour chaque dimension. Exemple:

```
VAR
    Table : Array [1..5,1..7] of Word;
```

Le choix de déclarer comme premier indice les lignes ou les colonnes est totalement arbitraire et est laissé à l'entière initiative du programmeur. Par contre ce choix devient définitif pour toute la durée d'utilisation du tableau. Autrement dit Table[5,7] est défini alors que Table [7,5] n'existe pas. Comme pour les autres tableaux on pourra utiliser chacune des valeurs contenues dans le tableau en indiquant les indices nécessaires.

Affectation	Table [1,5] := 1395; ou Table[1,1] := Table [5,5];
Lecture clavier	Readln (Table[4,7]);
Ecriture écran	Writeln (Table3,4]);[

Le passage d'un tableau de dimension quelconque en paramètre d'une procédure ou d'une fonction est toujours possible à condition d'avoir déclaré un type qui sera utilisé aussi bien pour la déclaration des variables que pour la déclaration des paramètres.

Exemple

Dans cet exemple, il est important de remarquer les doubles boucles FOR imbriquées qui permettent d'accéder à tous les éléments du tableau.

```
Program Tableau_2D;
Uses Crt;

Type
        T_Matrice = Array [1..5,1..5] of Byte;

Var
        Tableau_Test : T_Matrice;

Procedure Init (Var M : T_Matrice);
{ Cette procédure initialise toutes les valeurs de la matrice à une valeur aléatoire}
Var
        i,j : Byte;
begin
        For i:= 1 to 5 do
                For j:=1 to 5 do M[i,j] := Random (1000);
end;

Function Existe (N : Word; M : T_Matrice) : Boolean;
{ cette fonction renvoie la valeur VRAI si la valeur N est présente dans le tableau M }
Var
        i,j : byte;
        R : Boolean;
begin
        R := false;
        For i:=1 to 5 do
                For j:=1 to 5do
                        If M[i,j] = N then R := true;
        Existe := R;
end;

begin
        Randomize;
        Init (Tableua_Test);
        If Existe (5,Tableau_Test)
                then Writeln ('5 est présent dans le tableau initialisé')
                Else Writeln ('5 n"est pas présent dans le tableau initialisé');
end.
```

Exercices

Faire les exercices du chapitre : Les Tableaux.

LES FICHIERS

GENERALITES

Qu'est ce qu'un fichier?

Un fichier est un ensemble structuré d'informations. Cet ensemble est identifié par un nom et stocké sur une mémoire périphérique. cette mémoire est généralement un disque dur ou une disquette. En effet, toutes les données se trouvant dans la mémoire vive de l'ordinateur au moment de la coupure d'alimentation sont perdues. Un fichier permet donc de STOCKER de l'information en vue de la réutiliser lors de la remise en service de l'appareil. On est donc ammené ainsi à stocker, des textes, des graphismes, des fiches ou des programmes (en code source ou en code executable).

Il convient de distinguer deux aspects d'un fichier:

L'aspect logique : C'est le fichier vu du point de vue du programmeur. Les fiches y sont classées par ordre alphabétique, les lignes d'un textes sont placées les unes derrière les autres etc...

L'aspect physique : C'est le fichier vu du point de vue du système d'exploitation. Le fichier est organisé en secteurs sur les pistes du disque.

Pour faire communiquer ces deux points de vue, le système d'exploitation dispose d'un certain nombre de commandes qui permettent d'effectuer des lectures et des écritures dans le fichier. Ces commandes seront la plupart du temps déclenchées à partir d'un langage de programmation (TURBO-PASCAL ou autre) ou d'un logiciel de gestion de fichiers.

Actions sur les fichiers

On peut réaliser sur les fichiers un certain nombre d'actions qui sont, pour les principales:

Création	Donner un nom. Définir la structure.
Destruction	
Modification	ajout, suppression d'enregistrements.
Recherche	recherche d'une information dans le fichier
Visualiser	
Trier	
Fusionner	réunir/Séparer deux fichiers.
Imprimer	

Accès au contenu d'un fichier

Trois types d'accès sont possibles sur les fichiers informatiques.

Accés séquentiel

Dans ce type d'accès, on se place au début du fichier, et il est nécessaire de lire TOUS les enregistrements qui précédent le nième pour lire ce nième enregistrement. C'est le type même d'accès réservé aux fichiers stockés sur bande magnétique. Il est impossible d'insérer des fiches, seul l'ajout en fin de fichier est autorisé.

Accès direct

Chaque enregistrement porte un numéro. Il suffit d'invoquer ce numéro pour accéder à l'enregistrement désiré. La manipulation de ces fichiers est rapide, mais nécessite de connaitre le numéro d'enregistrement, c'est à dire sa place dans le fichier.

Accès indexé

C'est une des méthodes les plus fréquemment utilisées pour gérer les fichiers d'enregistrements. On utilise à la fois un fichier à accès direct pour stocker les données et une clé d'accès rangée dans un fichier d'index (en général à accès séquentiel).

TYPES DE FICHIERS

Fichiers Texte

C'est un fichier dans lequel on stocke, sous forme de codes ASCII, les caractères composant un texte. Ce texte pouvant être une oeuvre littéraire. Les données sont stockées dans des lignes qui sont de longueur variable et séparées par le couple de caractères CR-LF (Codes ASCII 13-10).

Il est à noter que Turbo-Pascal considère l'écran et le clavier comme des fichiers texte.

Fichiers Typés

Ce sont des fichiers destinés à stocker un nombre indéfini d'objets d'un même type. Entendons ici par objet un enregistrement. Cet enregistrement est d'un type donné Char, Word, Record[2]. Les différents enregistrements sont placés les uns à la suite des autres et sont repérables par leur numéro d'ordre. Le premier enregistrement stocké porte le numéro 0.

[2] Enregistrement pouvant rassembler différents types de données simples telles que chaine de caractères, nombres ou caractères.

Fichiers graphiques

Ces fichiers permettent de stocker des images ou des plans. Il existe un grand nombre de structures pour ces fichiers et leur gestion est complexe.

Fichiers exécuables

Ce sont des fichiers contenant des instructions en langage machine qui sont directement exécutables par la machine. Leur lecture et écriture est le plus généralement transparent pour l'utilisateur.

LOGICIELS DE GESTION DES FICHIERS

Avant de considérer les moyens de programmer la gestion des fichiers, il convient de préciser que des logiciels existent et permettent de gérer avec une très grande facilité et une très grande fiabilité des sommes importantes de données, et ce, dans différents domaines.

Traitement de Texte

Ce type de logiciel est spécialisé dans la gestion des fichiers textes. Les plus rudimentaires sont appelés EDITEURS de TEXTE, les plus sophistiqués TRAITEMENT DE TEXTE. Ces derniers permettent d'exploiter les imprimantes et d'obtenir des textes dont les caractères sont de différentes taille et style. Les plus récents permettent d'insérer des graphismes. Exemple : WORD, WORKS...

Au delà du traitement de texte, la mise en forme des documents destinés aux publications (Presses, Publicité...) sont classés sous le terme de P.A.O.[3] Exemple : PUBLISHER...

Gestionnaires de Fichiers

Ce sont des logiciels qui permettent de concevoir et tenir à jour des données rassemblées en fiches tels qu'un carnet d'adresses ou la liste des membres d'une association. Ces logiciels disposent de commandes permettant d'éditer sur imprimante les données sous forme de listes ou de fiche. Ces gestionnaires ne permettent pas (ou peu) de relations entre les différents fichiers constitués.

Gestionnaires de Bases de données relationnelles (S.G.B.D.).

Dans ce type de logiciel, le fichier n'est plus considéré comme isolé. Une base de données comporte en général de très nombreux fichiers. Ce qui caractèrise une base de données relationnelle c'est qu'Ils peuvent être mis en relation les uns avec les autres pour en exploiter le contenu. Les mettre automatiquement à jour ou croiser les données dans le but de faire des statistiques par exemple. Le travail le plus important dans ce

[3] PRESENTATION ASSISTEE PAR ORDINATEUR.

type de logiciel est de déterminer la structure de chaque fichier et de programmer les relations entre ces fichiers. Exemple : DBASE, PARADOX, ACCES...

Langages de programmation

C'est le système le plus rudimentaire pour la gestion des fichiers. Chaque accès à un fichier doit être programmé à l'aide de plusieurs instructions. En contre partie, le programmeur a toute latitude pour gérer les données.

PROGRAMMATION SUR LES FICHIERS

Les principales instructions qui permettent de gérer les fichiers en TP5 sont rassemblées dans le memento.

Création et utilisation d'un fichier typé

C'est le cas le plus courant, en effet on voudra créer un fichiers stockant des nombres réels, ou des fiches. Pour créer un fichier et remplir un tel fichier il faudra opérer de la façon suivante:

1. Déclarer un fichier logique. Cette déclaration, pour un fichier typé, se fait dans le paragraphe Var en utilisant le mot réservé File of suivi du type de données à stocker:

   FichierReel : **File of** Real;

2. Etablir un lien entre le fichier logique et le fichier physique en utilisant la procédure Assign:

   Assign (FichierReel,'A:\TURBO\DONNEES.DTA');

   qui associe le fichier logique FichierReel au fichier physique DONNEES.DTA qui se trouve dans le répertoire TURBO du lecteur A.

3. Créer ou Ouvrir le fichier. En effet, suivant la procédure utilisée RESET ou REWRITE le fichier ne sera pas traité de la même manière.

   RESET ouvre un fichier existant et place le pointeur de lecture sur la première donnée. (et provoque une erreur d'exécution si le fichier physique n'existe pas!)

   REWRITE ouvre un fichier vierge. Il le crée s'il n'existe pas ou l'ouvre et le vide complétement s'il existe déja.

4. Ecrire ou lire les données dans le fichier au moyen des procédure READ et WRITE.

   READ permet de lire une donnée et s'écrit en précisant en premier paramètre le fichier logique précédemment défini.

   READ (FichierReel, A);

   Cette instruction lit une valeur dans le fichier, la place dans la variable A et place le pointeur de lecture sur l'élément suivant du fichier. Il va de soi que la variable doit être du même type que les éléments constituants du fichiers.

   WRITE (FichierReel, A);

   cette instruction écrit, à l'endroit où se trouve le pointeur de fichier, le contenu de la variable A et avance le pointeur de fichier sur la position suivante d'écriture. A doit être du même type que les éléments constitutifs du fichier.

5. Fermer le fichier en utilisant la procédure CLOSE qui réalise les opérations de libération de mémoire et de cloture du fichier physique.

   CLOSE (FichierReel);

Exemple

Ce premier programme permet de créer un tableau des 200 entiers et de le stocker dans un fichier nommé NOMBRES.INT.

```
PROGRAM Sauve_Fichier;

USES Crt;

CONST
  Max = 200;

TYPE
  T_Tableau = ARRAY [1..Max] OF INTEGER;

VAR
  Table : T_Tableau;

PROCEDURE Initialisation (VAR T : T_Tableau);
VAR
  I : INTEGER;
BEGIN
  randomize;
```

```
  FOR i:=1 TO Max DO T[i] := random(1000);
END;

PROCEDURE Sauvegarde (T : T_Tableau);
VAR
  Fichier : FILE OF INTEGER;
  i       : WORD;
BEGIN
  Assign (Fichier,'NOMBRES.INT');
  Rewrite (Fichier);
  FOR i:=1 TO Max DO
  BEGIN
    Writeln (T[i]);
    Write (Fichier,T[i]);
  END;
  Close (Fichier);
END;

BEGIN
  ClrScr;
  Initialisation (Table);
  Sauvegarde (Table);
END.
```

Ce deuxième programme permet de relire les données stockées et de les replacer dans un tableau.

```
PROGRAM Charge_Fichier;

USES Crt;

CONST
  Max = 200;

TYPE
  T_Tableau = ARRAY [1..Max] OF INTEGER;

VAR
  Table : T_Tableau;

PROCEDURE Chargement (VAR T : T_Tableau);
VAR
  Fichier : FILE OF INTEGER;
  i       : INTEGER;
BEGIN
  Assign (Fichier,'NOMBRES.INT');
  Reset  (Fichier);
  I := 0;
```

```
  WHILE NOT Eof(Fichier) DO
  BEGIN
    Inc (I);
    Read (Fichier,T[i]);
    Writeln (T[i]);
  END;
  Close (Fichier);
END;

BEGIN
  ClrScr;
  Chargement (Table);
END.
```

Exercices

Faire les exercices du chapitre : Les fichiers.

Création et utilisation d'un fichier texte

La déclaration d'un fichier texte n'est pas la même que pour un fichier typé. On utilisera le mot réservé TEXT.

```
MonTexte : TEXT;
```

On l'assignera et on l'ouvrira de la même façon qu'un autre fichier, mais la lecture se faisant ligne par ligne, on utilisera les procédures READLN et WRITELN pour y lire et y écrire.

Exemple

Cet exemple permet de stocker 5 phrases dans un fichier texte. Remarquez que vous pourrez ouvrir ce fichier directement à partir de l'éditeur de TurboPascal et que vous pourrez le lire sans problème. C'est normal puisque vos programmes Pascal sont stockés sous forme de fichiers Texte.

```
Program TestFichier;
Uses Crt;

Var
      Buffer : Text;

Begin
      Assign (Buffer,'A:MONTEXTE.ASC');
      Rewrite (Buffer);
      For i:=1 to 5 do
```

```
        begin
                Writeln ('Donner une phrase à stocher dans le fichier : ');
                Readln (Phrase);
                Writeln (Buffer, Phrase);
        end;
        Close (Buffer);
end.
```

TABLE DES MATIERES

INDEX