**République Algérienne Démocratique et Populaire**

**Ministère de l'enseignement supérieur et de la recherche scientifique**

**Université de Mostaganem Abdelhamid Ibn Badis**

**Faculté des Sciences et de la Technologie**

**Département des sciences et techniques**

# Manuel de travaux pratiques

# Module : Informatique

**Elaboré par :**

**Dr. ROUBA Baroudi**

**Maître de conférences « A »**

**Expertisé par :**

**Pr. MERAH Mustapha**

**Dr. Laredj Mohammed Adnane**

# Résumé

La programmation constitue l'un des moyens les plus importants pour automatiser le traitement de l'information. En informatique, ils existent plusieurs langages de programmation. Le langage Pascal représente l'un des langages de programmation les plus utilisés dans le domaine de l'enseignement. Ce polycopié explique d'une manière détaillée comment implémenter un programme informatique en utilisant le langage Pascal et comment l'exécuter sur un ordinateur.



# Mots clés

Langage Pascal, Programme, Algorithme, Compiler, Exécuter.

# Avant-propos

Ce manuel de travaux pratiques est destiné aux étudiants de la 1ère année du domaine des sciences et technologies. Il représente un complément du cours d'Informatique où sont dispensés les principaux concepts du langage algorithmique.

Ce manuel est consacré à l'apprentissage du langage Pascal sous l'éditeur « Turbo Pascal pour Windows » version 1.5. De ce fait, toutes les étapes d'installation de cet outil sont détaillées au niveau de la première séance.

Pour maitriser les différents concepts liés à la programmation en langage Pascal, le présent manuel est structuré en séances. Chaque séance introduit un nouveau concept et la fin de chaque séance l'étudiant sera en mesure d'utiliser le concept présenté dans un programme.

Afin de faciliter l'assimilation des concepts abordés, des exercices d'application sont proposés et leurs solutions sont détaillées.

# Sommaire

# 1. Introduction à la programmation en langage Pascal

## 1.1.Définition d'un algorithme

L'origine du mot algorithme est liée au nom du célèbre mathématicien arabe Al Khawarismi. L'algorithme peut être défini comme étant le procédé à suivre, étapes par étapes, pour résoudre un problème donné. Les étapes à suivre pour résoudre une équation peuvent être considérées comme un algorithme.

Exemple : On considère l'équation $5x + 9 = 2x - 7$

Suivre les instructions ci-contre :

- Retrancher 9 dans les deux membres
- retrancher 2x dans les deux membres
- diviser par 3 les deux membres
- écrire la solution de l'équation

Un algorithme peut être défini comme étant une suite d'instructions élémentaires qui s'appliquent dans un ordre déterminé à des données et qui fournissent en un nombre fini d'étapes des résultats.

## 1.2.Définition d'un programme

Un programme est la traduction d'un algorithme dans un langage de programmation compréhensible par la machine.

## 1.3.Définition d'un langage de programmation

L'ordinateur ne fait qu'exécuter ce que le programmeur lui demande de faire. Pour ce faire, le programmeur doit formuler les ordres dans un langage compréhensible par l'ordinateur mais du fait que ce dernier ne comprend que des ordres codés en binaire, des langages dit évolués ont été créés pour faciliter la programmation.

Il existe plusieurs langages de programmation tels que le langage C, fortran, Java...etc. Le présent manuscrit traite le langage Pascal.

## 1.4.Le langage Pascal

Le langage Pascal a été créé au début des années 1970 par Niklaus WIRTH à l'école polytechnique de ZURICH en Suisse. Le nom Pascal a été donné à ce langage à l'honneur du mathématicien français Blaise PASCAL.

C'est l'un des langages les plus facile à utiliser à cause de la simplicité de ces instructions qui sont proches du la langue anglaise.

## 1.5.Le Turbo Pascal pour Windows (TPW)

Plusieurs compilateurs et éditeurs du langage Pascal sont disponibles. Dans ce manuel, nous allons utiliser le ***Turbo Pascal 1.5 pour Windows*** téléchargeable gratuitement (après inscription) depuis l'url suivante:

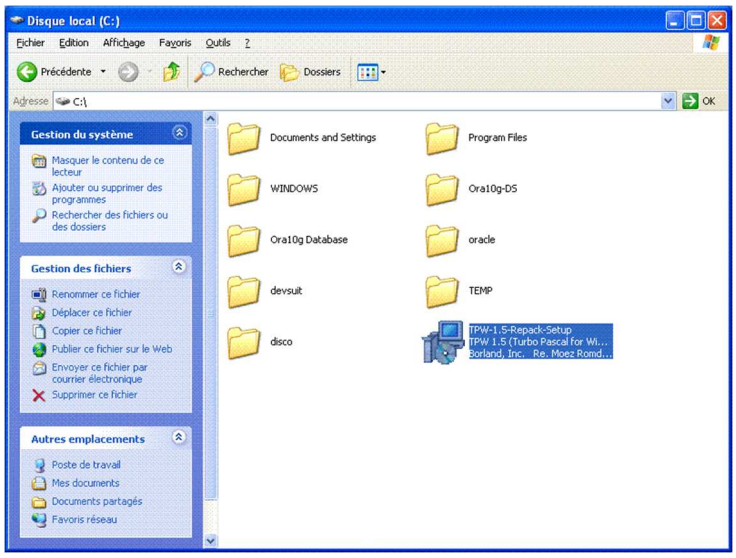
https://pascal.developpez.com/telecharger/telecharger/id/5685/Turbo-Pascal-for-Windows-1-5-Repack

***Il est à noter qu'il s'agit d'une application qui s'exécute uniquement sur une version 32 bits de Windows.***

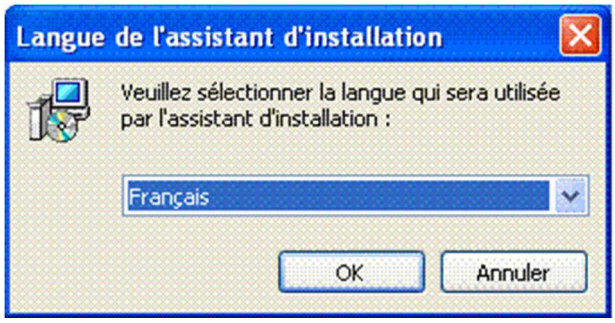
## 1.6.Installation de Turbo Pascal pour Windows

Pour installer le Turbo Pascal 1.5 pour Windows, il faut suivre les étapes ci-dessous.

1. Double-cliquez sur le fichier d'installation "TPW-1.5-Repack-Setup"

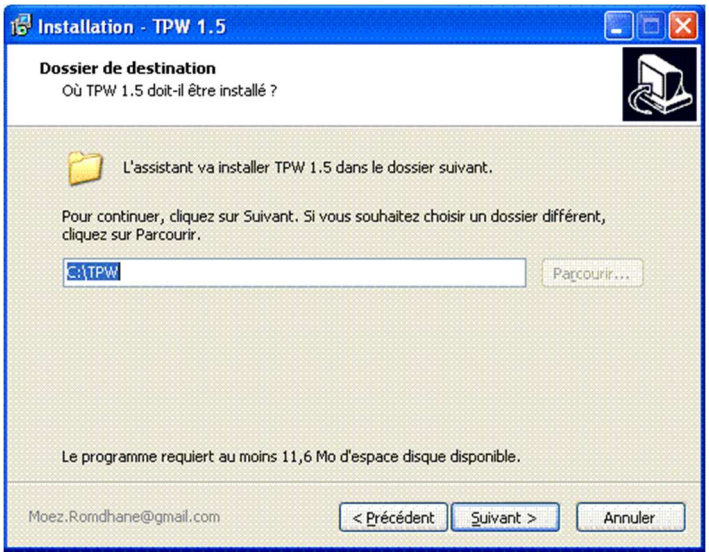



2. Choisir la langue de l'assistant d'installation



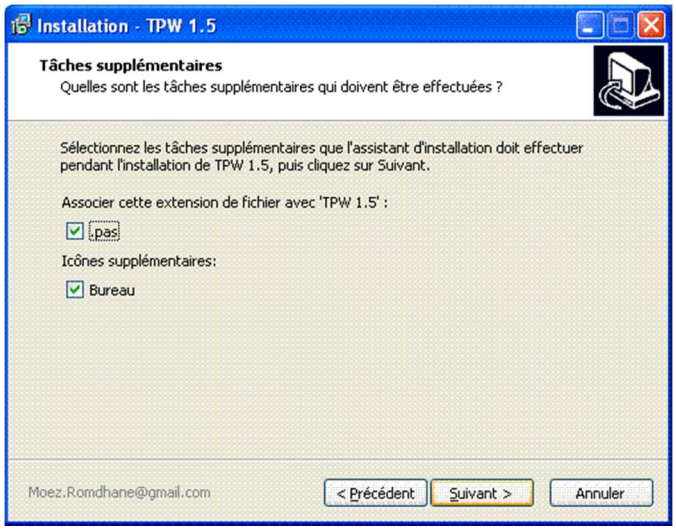

3. Cliquer sur le bouton "Suivant" sur la fenêtre "bienvenue" de l'installation


4. Choisir le dossier de destination ou le Turbo Pascal sera installé. L'assistant vous propose automatiquement le dossier "C:\TPW". Il est possible de choisir un autre dossier. Une fois le dossier choisi cliquer sur "Suivant".

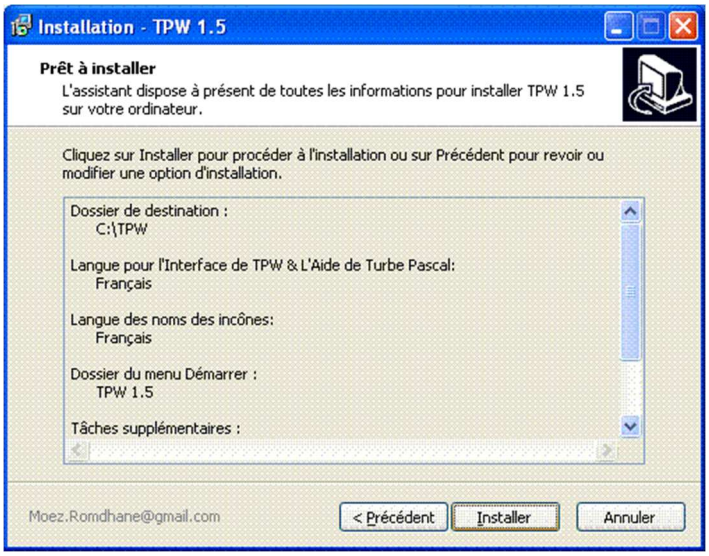



5. Choisir la langue à utiliser pour l'affichage de l'interface du Turbo Pascal, de l'aide et des icônes puis cliquer sur "Suivant"


6. Choisir les taches complémentaires que l'assistant doit exécuter à la fin de l'installation puis cliquer sur "Suivant".

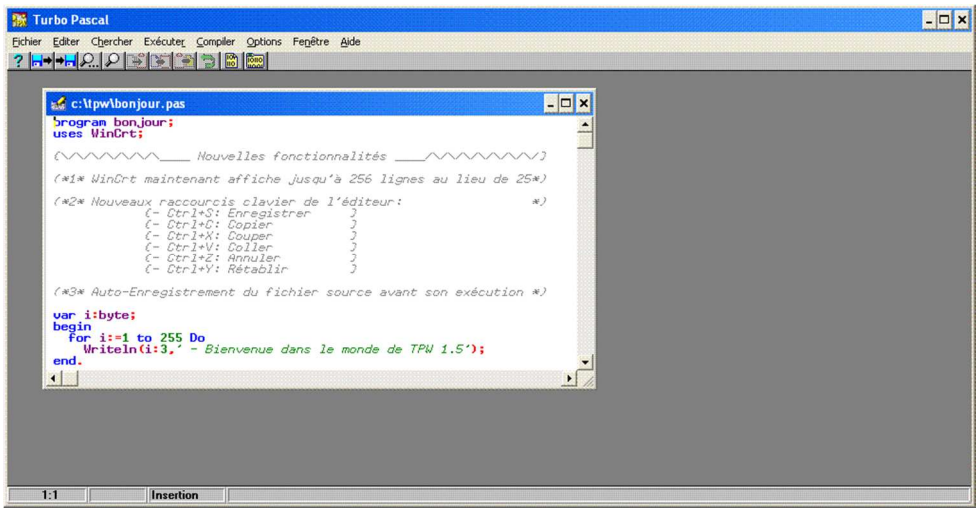



7. Lancer l'installation en cliquant sur le bouton "Installer"




8. Cliquer sur le bouton "Fin" pour terminer l'installation.


9. A la fin de la procédure d'installation. Le Turbo Pascal est lancé affichant le code d'un un programme nommé "bonjour" stockés dans fichier nommé "bonjour.pas"




## 1.7.Structure d'un programme en Pascal

Un programme Pascal est composé de 3 sections : la section d'en-tête, la section déclarative et la section exécutable.

### 1. L'en-tête

Cette section sert à identifier la tâche à résoudre. Tout programme en Pascal doit commencer par le mot clé ***program*** suivi par le nom du programme qui se termine par un point-virgule « ; ».

### 2. La section déclarative

Cette section est dédiée à la déclaration des différents objets utilisés dans le programme tel que les unités, les variables et les constantes.

### 3. La section exécutable

Cette section contient les instructions à exécuter. Elle délimitée par les mots clés ***begin*** et ***end.***

# 2. TP N°1- Se familiariser avec Turbo Pascal pour Windows-

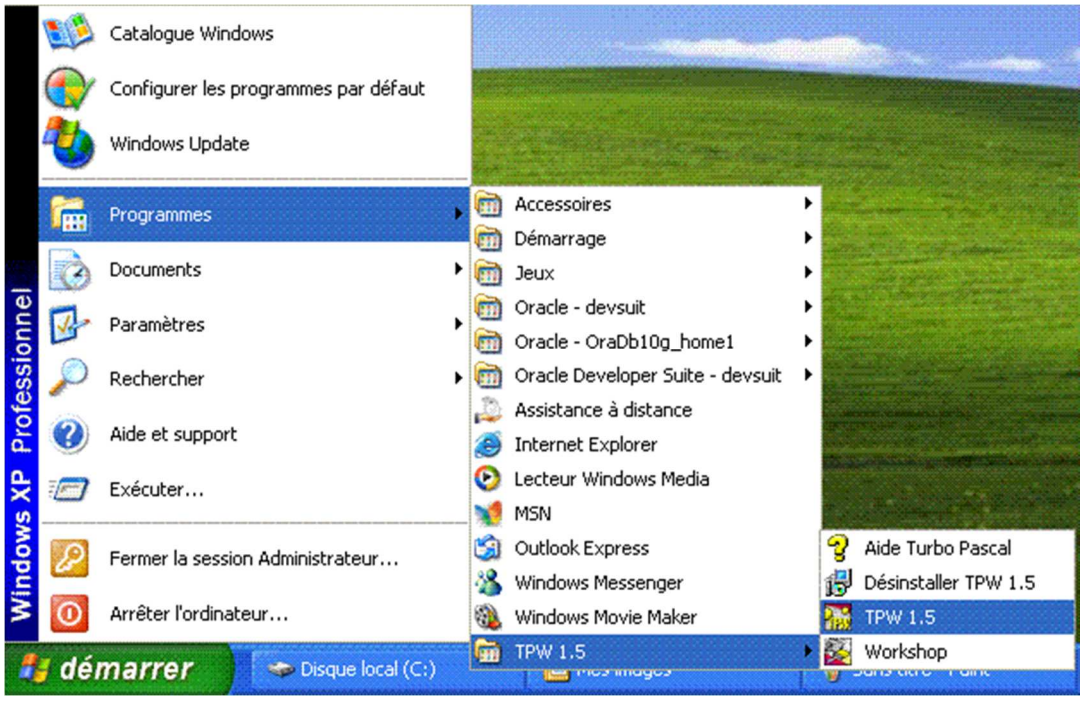
## 2.1.Objectifs

A la fin de cette séance, l'étudiant sera capable de :

- Lancer le logiciel Turbo Pascal pour Windows (TPW).
- Créer un nouveau fichier de programme.
- Editer un fichier de programme.
- Enregistrer un fichier de programme.
- Ouvrir un programme existant.
- Fermer un fichier.
- Quitter l'éditeur TPW.

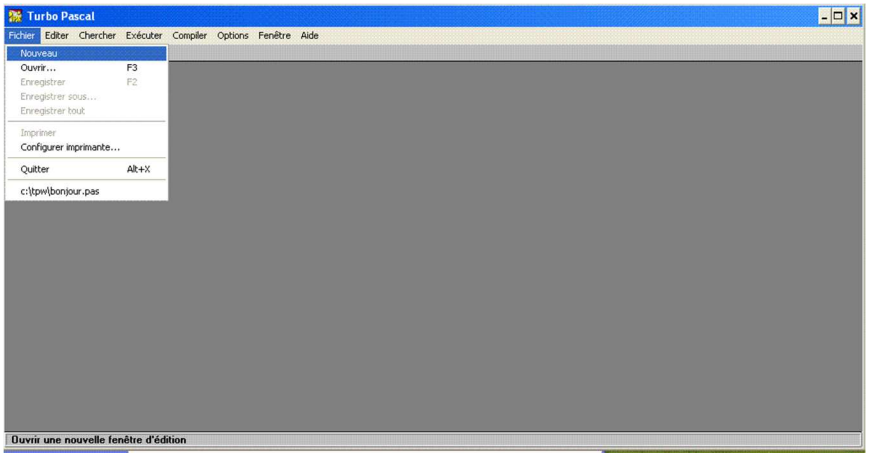
## 2.2.Lancement de TPW

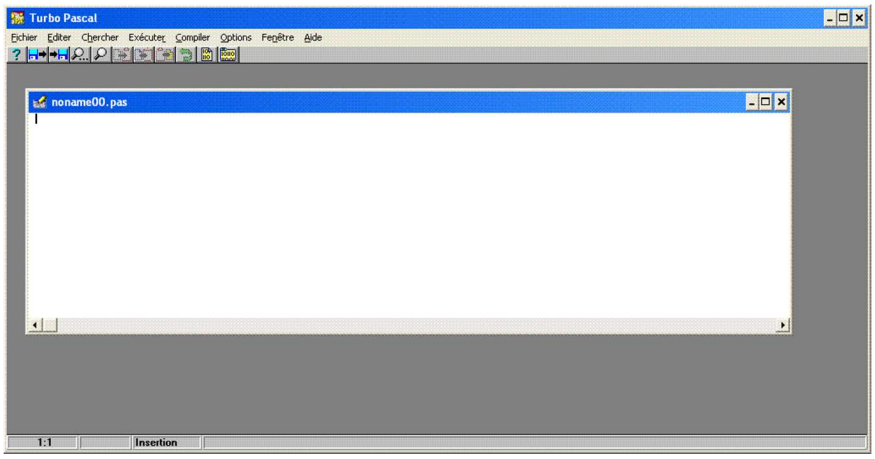
Pour lancer TPW, il faut cliquer sur le menu démarrer ->programmes->TPW 1.5->TPW 1.5 .




## 2.3.Création d'un nouveau fichier

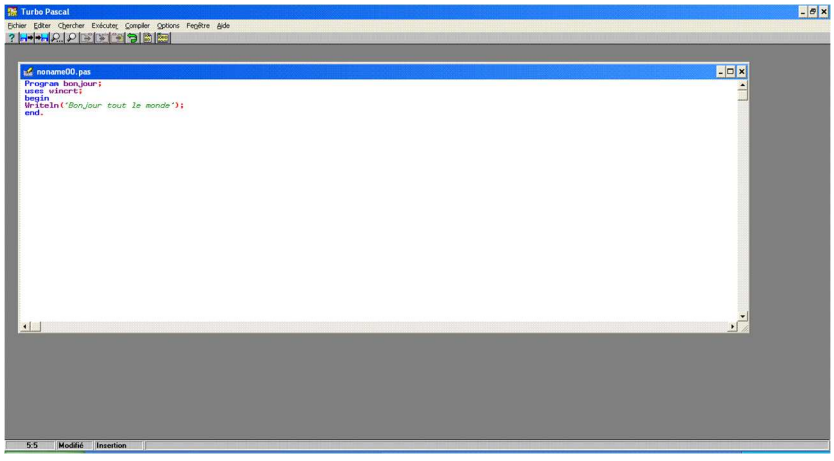
Pour créer un programme il faut d'abord créer le fichier sur lequel ce programme sera écrit. Pour créer un nouveau fichier, cliquer sur le menu Fichier->Nouveau. Une nouvelle fenêtre s'affiche portant le nom "noname00.pas".




## 2.4.Editer un fichier de programme

Pour créer un programme, il suffit d'éditer le fichier créé en écrivant les instructions du programme.

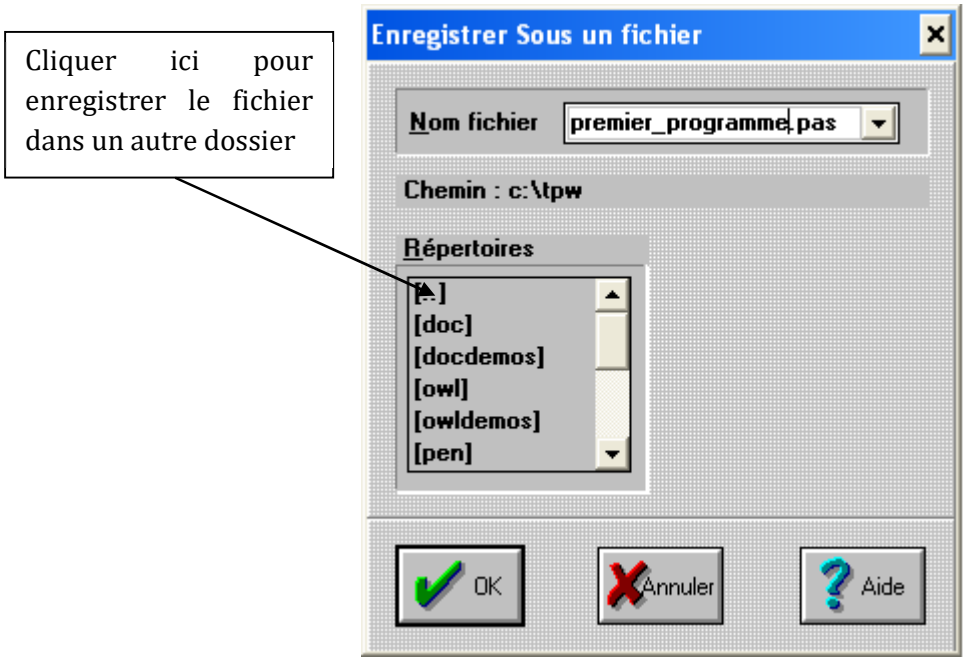



## 2.5.Enregistrer un fichier de programme

Pour enregistrer un fichier, utiliser le menu Fichier->Enregistrer ou le menu Fichier->Enregistrer sous. Une fenêtre s'affiche pour indiquer le nom du fichier et son emplacement.

L'emplacement par défaut est le dossier **"C:\TPW".** Pour enregistrer le fichier dans un autre emplacement, il suffit de cliquer sur **[..]** dans la zone "**Répertoire"** de la fenêtre.

Taper le nom du fichier dans la zone indiquée puis cliquer sur le bouton **"OK"**

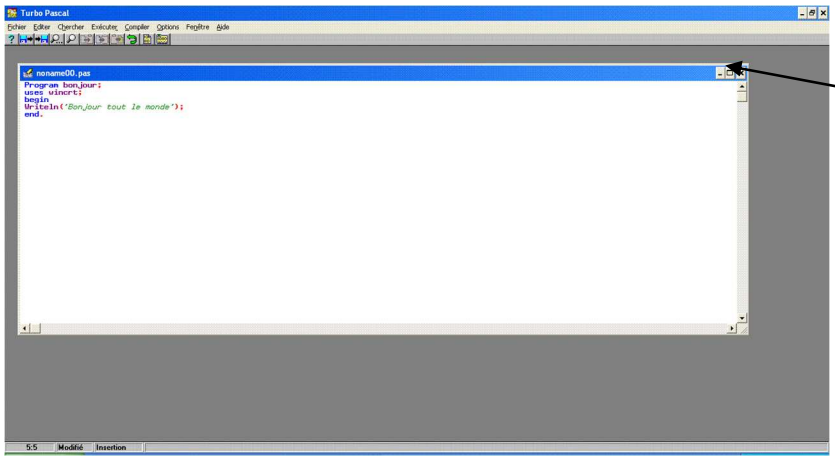



## 2.6.Ouvrir un programme existant

Pour ouvrir un programme existant, il suffit de choisir le menu Fichier-> Ouvrir. Dans la fenêtre qui s'affiche cliquer sur le nom du fichier à ouvrir puis cliquer sur **"OK".**

## 2.7.Fermer un fichier de programme

Pour fermer un fichier de programme, il suffit de cliquer sur la croix qui se trouve en haut à droite de la fenêtre du fichier




## 2.8.Quitter l'éditeur TPW

Pour quitter l'éditeur TPW, il faut cliquer sur le menu Fichier->Quitter.


## 2.9.Exercice d'application

1. Lancer l'éditeur TPW et créer un nouveau fichier.
2. Saisir le texte suivant dans le fichier

   ***Program tp1***

   ***begin***

   ***writeln('Salut tout le monde);***

   ***end.***
3. Enregistrer le fichier sous le nom "tp1.pas".
4. Fermer le fichier "tp1.pas".
5. Quitter l'éditeur TPW.

6. Relancer l'éditeur TPW puis ouvrir le fichier "tp1.pas".

# 3. TP N°2- Editer, compiler et exécuter un programme-

## 3.1.Objectifs

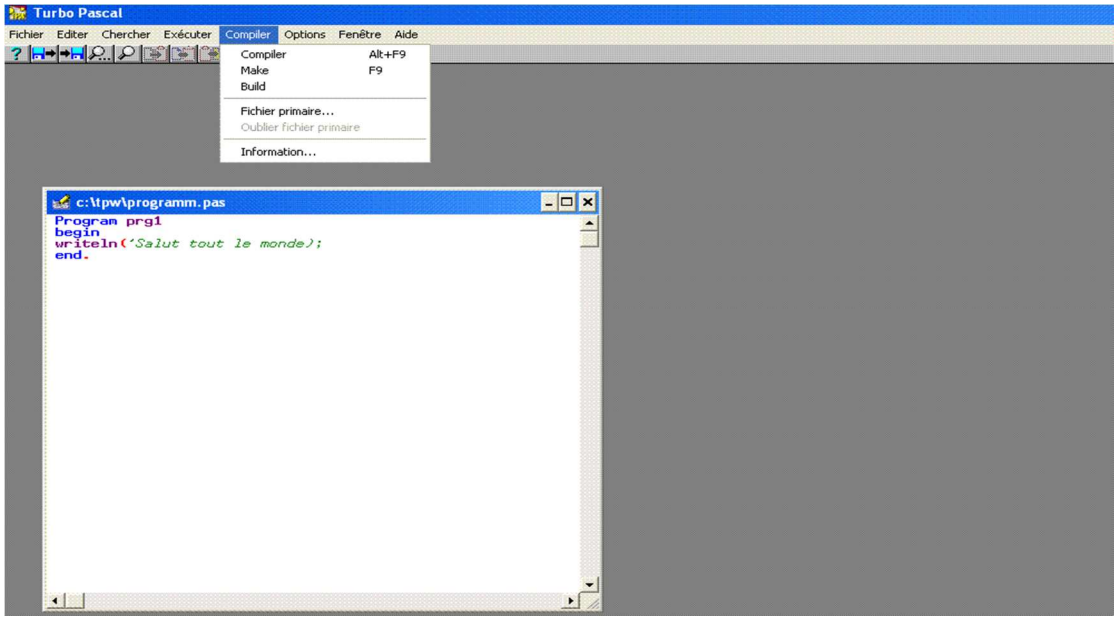
A la fin de cette séance, l'étudiant sera capable de :

- Compiler un programme.
- Corriger des erreurs de compilation.
- Exécuter un programme.

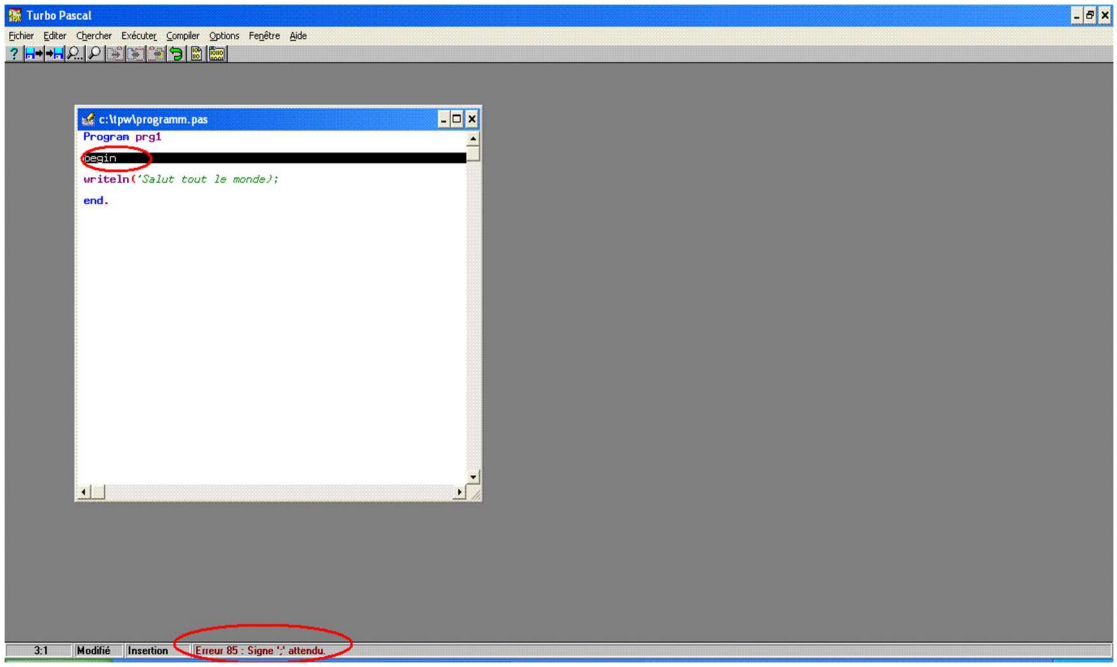
## 3.2.Compiler un programme

La compilation est le processus qui permet à la transformation d'un programme écrit dans un langage donné (dans notre cas c'est le langage Pascal) en un programme exécutable par l'ordinateur. Pour que cette transformation soit possible, il faut que le programme soit écrit correctement dans la syntaxe du langage utilisé. En d'autres termes, pour pouvoir exécuter le programme il faut s'assurer qu'il ne contient aucune erreur de syntaxe.

Pour compiler un programme en TPW, il faut cliquer sur le menu Compiler->Compiler.




Lors de la compilation les erreurs sont repérées par le curseur clignotant sur la ligne surlignée. Le code de l'erreur et sa description sont affichés en barre d'état.

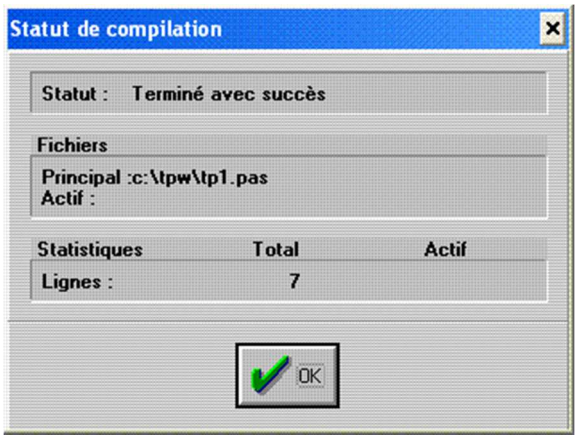



## 3.3.Corriger les erreurs de compilation

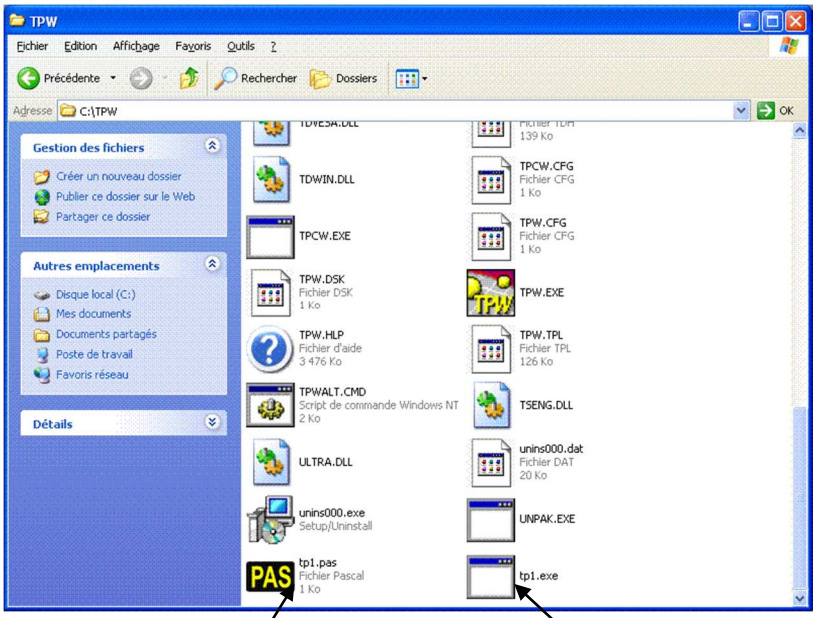
Pour corriger les erreurs de compilation, il faut lire la description de l'erreur affichée en barre d'état. L'exemple de la figure 18 indique que le manque d'un point-virgule ***"Erreur 85: Signe ";" attendu"*** . Pour corriger cette erreur, il faut repérer le manque du signe ";" dans la ligne surlignée ou la ligne qui la précède. Dans ce cas, la ligne surlignée contient le mot clé "Begin" qui ne doit pas être suivi par un point-virgule. La source de l'erreur est donc sur la ligne qui précédente qui contient "Program tp1". Cette dernière doit être obligatoirement suivie par un point-virgule.

Une fois l'erreur corrigée, il est nécessaire de compiler le programme de nouveau pour s'assurer que le programme ne contient plus d'erreurs.

Si le programme ne contient aucune erreur. Le TPW affiche une boite de dialogue indiquant que la procédure de compilation s'est terminée avec succès et que le fichier exécutable du programme a été créé dans le même chemin que le fichier source du programme.



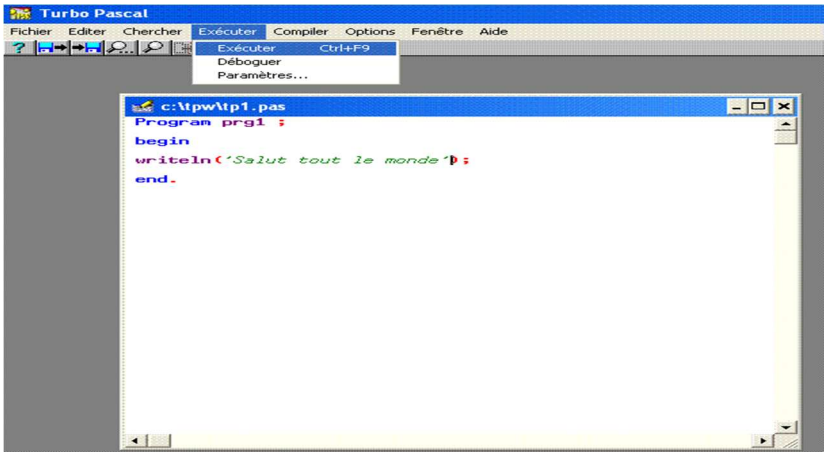

Le fichier exécutable aura le même nom que le fichier source mais avec l'extension ".exe". C'est à dire si le fichier source du programme est nommé "tp1.pas" son fichier exécutable aura le nom "tp1.exe"



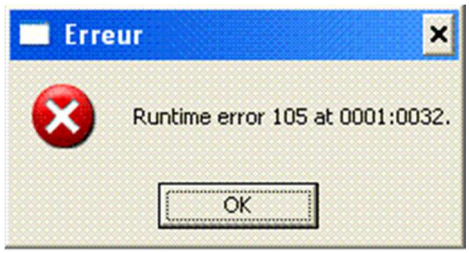

Fichier source du programme Fichier exécutable du programme

## 3.4.Exécuter un programme

Pour exécuter un programme, il faut cliquer sur le menu Exécuter->Exécuter. En réponse, la fenêtre d'exécution s'affiche à l'écran.




Si une erreur se produit au cours de l'exécution une boîte de dialogue indique le code d'erreur et l'adresse mémoire de l'erreur.




## 3.5. Exercice d'application

1. Lancer TPW.
2. Ouvrir le fichier "tp1.pas" créé durant la séance précédente.
3. Compiler et corriger les erreurs du programme
4. Exécuter le programme.

## 3.6.Corrigé de l'exercice d'application

1. **Lancer TPW :**

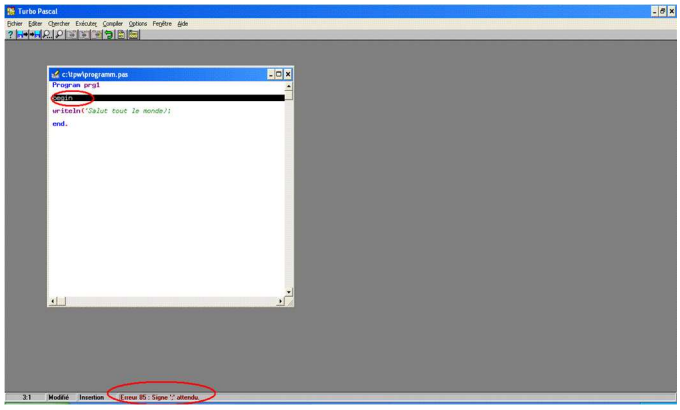
   Pour lancer TPW, il faut cliquer sur le menu démarrer->tous les programmes->TPW 1.5>TPW 1.5

2. **Ouvrir le fichier "tp1.pas" :**

   Pour ouvrir le fichier "tp1.pas", cliquer sur le menu Fichier-> Ouvrir, puis sélectionner le fichier "tp1.pas" et cliquer sur ok.

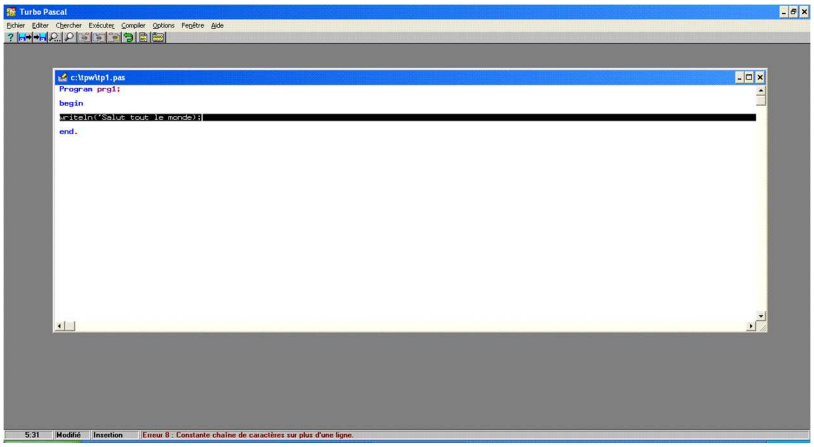

3. **Compiler et corriger les erreurs du programme :**

   Pour compiler le fichier, cliquer sur Compiler->Compiler. Le TPW indique une première sur la ligne contenant le mot clé "Begin". L'erreur est codée "Erreur 85", avec la description "Signe ';' attendu.



   Le compilateur indique le manque du signe ";" sur la ligne "Begin" ou celle qui la précède. Puisque le mot clé "Begin" ne doit pas être suivi par le signe ';' alors c'est la première ligne qui doit être complété par ce signe.

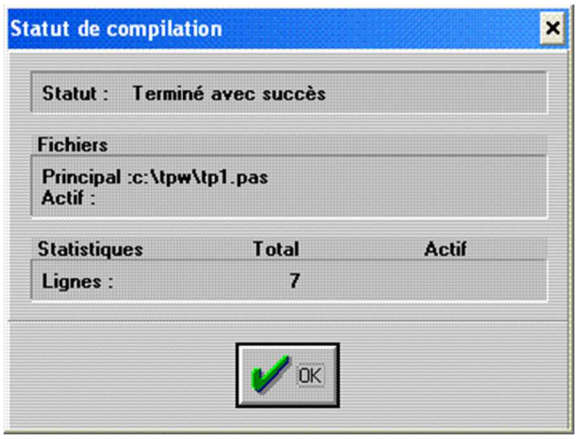
   Ajouter le signe ';' à la fin de la première ligne : **Program tp1;** puis compiler le programme de nouveau.




Cette fois-ci, le TPW indique une erreur sur la ligne ***writeln('salut tout le monde);*** . L'erreur est codée "Erreur 8" avec la description ***"Constante chaîne de caractères sur plus d'une ligne"*** .

La source de l'erreur est due au fait que le message de l'instruction writeln doit être inclut entre deux apostrophes (au début et à la fin). L'erreur est donc due à l'absence de l'apostrophe de la fin. Pour corriger l'erreur il suffit d'ajouter une apostrophe après le mot "monde" et avant la parenthèse fermante. L'instruction corrigée est : ***writeln('salut tout le monde');***

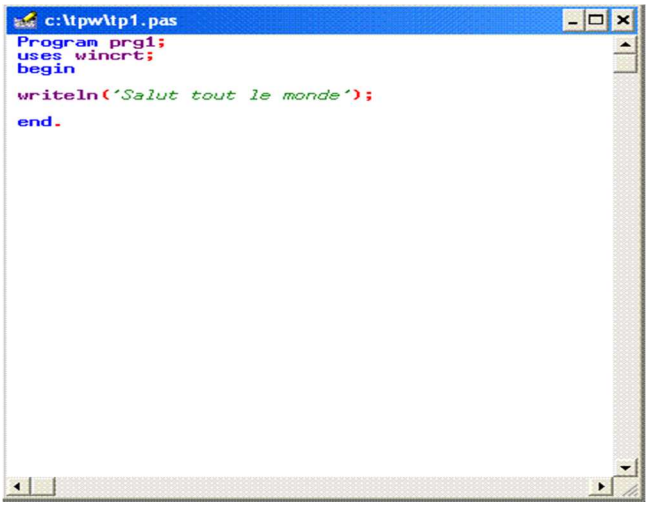
Recompiler le programme de nouveau. Cette fois ci le TPW affiche un message de succès.



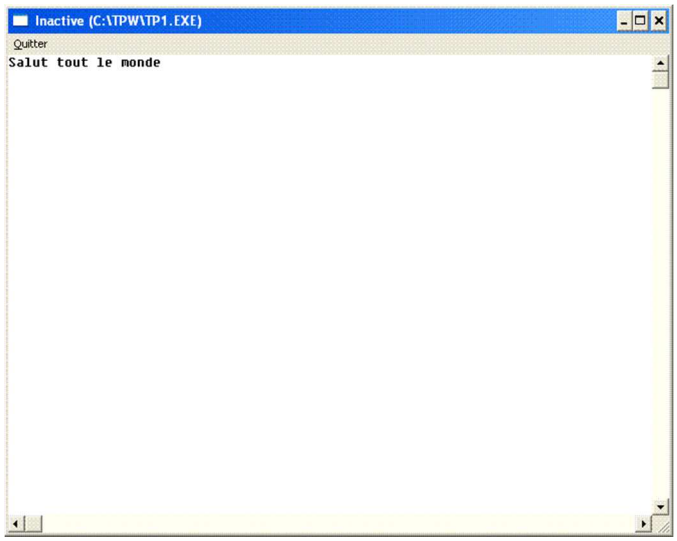

4. **Exécuter le programme :**

Pour exécuter le programme, cliquer sur le menu Exécuter->Exécuter. Le programme indique une erreur d'exécution.


Cette erreur est due à l'absence d'une librairie nécessaire à l'exécution du programme sous Windows. La libraire est nommée **« wincrt».** Pour faire appel à cette libraire il faut ajouter la ligne « **uses wincrt ; »** au programme. Cette instruction doit toujours être incluse dans les programmes sur la 2ème ligne (la ligne qui suit celle du nom du programme).




En exécutant le programme de nouveau, le message "Salut tout le monde " s'affiche sur la fenêtre d'exécution.


Inactive (C:\TPW\TP1.EXE)
Quitter
Salut tout le monde

# 4. TP N°3-Les types de données standards (prédéfinis)-

## 4.1.Objectifs

A la fin de cette séance, l'étudiant sera capable de :

- Décrire la structure détaillée d'un programme Pascal.
- Déclarer des constantes
- Déclarer des variables
- Identifier les types de données supportés par TPW.
- Identifier les fonctions standards applicables sur chaque type.

## 4.2.Syntaxe de la structure détaillée d'un programme Pascal

Comme il a été signalé précédemment, un programme en Pascal est organisé en trois sections. La section d'en-tête, la section déclarative et la section exécutable. Ci-dessous la syntaxe de la structure détaillée d'un programme.

| | | |
|---|---|---|
| **Program** nom_programme ; | {Nom du programme} | Section d'en-tête |
| **Uses** wincrt ; | {Librairie nécessaire à l'exécution} | Section déclarative |
| **Const** identificateur_constante= valeur ; | {Déclaration des constantes si elles existent} | |
| **Type** identificateur_type=description ; | {Déclaration des types s'ils existent} | |
| **Var** identificateur_variable1: type1;<br>Identificateur_variable2: type2;<br>.<br>.<br>. | {Déclaration des variables} | |
| **Begin** | {Début de la section exécutable} | Section exécutable |
| Instruction_1;<br>Instruction_2;<br>Instruction_3;<br>.<br>.<br>.<br>Instruction_n; | | |
| **End.** | {Fin du programme} | |

### Règles à respecter

- Les déclarations sont séparées par un « ; »
- Les instructions sont séparées par un « ; »
- Le « ***End »*** final est terminé par un point. Tout ce qui suivra sera ignoré par le compilateur.
- Les accolades représentent des marqueurs de commentaires. Tout ce qui entre accolades est ignoré par le compilateur.

## 4.3.Déclaration des constantes

Une constante est un objet dont la valeur est fixe tout au long du programme et pour chaque exécution. La déclaration se fait avec le mot réservé CONST :

**CONST**

**identificateur_constante=valeur;**

**Exemple**

```
Const
pi=3.14;
g=9.8;
ville='Mostaganem';
```

Le code de l'exemple permet de déclarer trois constantes. Les deux premières, pi et g numériques dont les valeurs respectifs sont 3.14 et 9.8. Le troisième constant est une chaine de caractères dont la valeur est 'Mostaganem'.

## 4.4.Déclaration des variables

Une variable est un objet destiné à recevoir une valeur qui peut être utilisée suivant les instructions du programme. Les variables peuvent avoir différents types suivant les valeurs qui seront stockées dedans. La déclaration des variables se fait par le mot clé **VAR**

**VAR**

**nom_variable : type;**

Il est possible de déclarer plusieurs variables de même type en une seule ligne en les séparant par des virgules.

## 4.5.Les types de données standards

### 1. Le type Integer

Ce type représente les nombres entiers compris entre -32768 et +32767. Plusieurs variantes de ce type existent.

| Type | Domaine |
|---|---|
| Shortint | Entre -128 et 127 |
| Longint | Entre -2147483648 et 2147483647 |
| Byte | Entre 0 et 255 |
| Word | Entre 0 et 65535 |

Les opérations applicables sur le type ***Integer*** sont :

| Opérateur | Opération |
|---|---|
| + | Addition |
| - | Soustraction |
| * | Multiplication |
| / | Division |
| Div | Division entière |
| Mod | Reste de la division entière (modulo) |

## 2. Le type Real

Le type réel prend ses valeurs dans un sous ensemble des réels. On utilise le point comme marque de décimale.

Les opérations applicables sur le type ***Real*** sont :

| Opérateur | Opération |
|---|---|
| + | Addition |
| - | Soustraction |
| * | Multiplication |
| / | Division |

## 3. Le type Char

Comporte les lettres, les chiffres, les signes de ponctuations (. etc.), les symboles utilisés en tant qu'opérateurs (+, *, -, /, <, =, etc.), les caractères spéciaux (%, @, $, &, ...). Un caractère est encadré par 2 apostrophes.

## 4. Le type String

Représente une suite de caractères. Une chaine est encadrée par 2 apostrophes. Toute apostrophe figurant dans une chaine doit être doublée.

## 5. Le type Boolean

Une variable de type ***Boolean*** peut prendre deux valeurs possibles soit ***vrai*** soit ***faux***.

Les opérations applicables sur le type ***Boolean*** sont :

| **Opérateur** | **Opération** |
|---|---|
| and | La conjonction |
| Or | La disjonction |
| Not | La négation |
| Xor | Le OU exclusif |

Les valeurs de type ***Boolean*** peuvent être générées par les opérateurs de comparaison suivants :

| **Opérateur** | **Signification** |
|---|---|
| > | Supérieur |
| >= | Supérieur ou égal |
| < | Inférieur |
| <= | Inférieur ou égal |
| = | Egal |
| <> | Différent |

## 6. Les fonctions standards (predefinies)

Le langage Pascal propose un ensemble de fonctions utilisables pour chaque type de donnée. Une fonction standard permet de calculer automatiquement une valeur. La valeur dépend du type de donnée sur lequel la fonction est applicable.

Dans le tableau ci-dessous sont présentées les principales fonctions utilisées pour chaque type de données.

| Fonction | Description | Type de données en entrée | Type de données du résultat |
|---|---|---|---|
| ABS | La valeur absolue d'un nombre entier. | Integer/real | Integer/real |
| SQR | Le carré d'un nombre. | Integer/Real | Integer/Real |
| SQRT | La racine carrée d'un nombre. | Integer/Real | Real |
| EXP | L'exponentiel d'un nombre | Real | Real |
| LN | Le logarithme népérien | Real | Real |
| SIN | Le sinus d'un nombre | Real | Real |
| COS | Le cosinus d'un nombre | Real | Real |
| ARCTAN | L'arctangeante d'un nombre | Real | Real |
| ROUND | L'entier le plus proche d'un nombre réel | Real | Real |
| Length | La longueur (le nombre de caractères) d'une chaîne de caractères | String | Integer |

**Exemples :**

- ABS(-5) permet de calculer la valeur absolue du nombre -5 qui vaut 5.
- SQR(3) permet de calculer le carré du nombre 3 qui est égale à 9.
- SQRT(25) permet de calculer la racine carrée du nombre 25 qui vaut 5.
- ROUND(26.9) retourne la valeur 27 qui représente l'entier le plus proche du nombre 26.9
- Length('Bonjour') retourne le nombre 7 car le chaîne de caractères *Bonjour* contient 7 caractères.

## 7. Les priorités entre les opérateurs

Les expressions arithmétiques et/ou logiques non-parenthésées doivent être évaluées en respectant l'ordre de priorité entre les opérateurs. Les opérateurs sont ordonnés comme suit :

**Priorité 1 :** Not.

**Priorité 2 :** *, /, Mod, Div, And.

**Priorité 3 :** +, -, Or, Xor.

**Priorité 4 :** =, <, >, <=, >=, <>.

En absence de parenthèses, les opérateurs du même ordre de priorité sont évalués de gauche à droite.

**Exemples**

L'expression 2*A+3-B est évaluée comme suit:

- 2*A car l'opérateur * est le plus prioritaire.
- 2*A+3 car les opérateurs + et - ont le même ordre de priorité donc on commence par celui qui se trouve le plus à gauche.
- 2*A+3-B

Pour forcer un ordre bien précis, il faut utiliser les parenthèses.

## 8. Exercice d'application

1. Créer un nouveau programme nommé "tp2.pas".
2. nommer le programme ***cercle***.
3. Déclarer une constante nommée Pi dont la valeur est 3.14.
4. Déclarer trois variables nommées respectivement R, P, S de type réel.
5. Enregistrer le fichier du programme.

## 9. Corrigé de l'exercice d'application

Le programme se présente comme suit

```
program cercle;
uses wincrt;
const
pi=3.14;
var
R, P, S : real;
Begin


end.
```

Le programme sera complété au cours des exercices des prochaines séances.

# 5. TP N°4-Les instructions élémentaires-

## 5.1.Objectifs

A la fin de cette séance, l'étudiant sera capable d'utiliser :

- L'instruction d'affectation (***:=***).
- Les instructions d'écriture ***write*** et ***writeln***.
- Les instructions de lecture ***read*** et ***readln***.

## 5.2.L'affectation

L'affectation permet de stocker, dans une variable, une valeur décrite par une expression. En Pascal, l'affectation est symbolisée par le signe (:=). La mise d'une valeur dans une variable peut se faire directement ou sous la forme d'un calcul.

**Syntaxe**

***A:=E*** **;** où E représente une expression.

**Exemples**

- L'instruction ***A:=5*** permet d'affecter directement la valeur 5 à la variable A.
- L'instruction ***A:=B*C*** permet d'affecter à la variable A le résultat de la multiplication de B par C.

L'affectation n'est valide que si le type de l'expression est compatible avec le type de la variable. Si A est une variable de type ***Integer***, alors l'affectation A:='bonjour' n'est pas valide. De même si A est une variable de type entier et B est une variable de type ***Real*** alors l'affectation ***A:=B*** n'est pas valide. Par contre l'affectation ***B:=A*** est valide car elle respecte l'inclusion du type ***Integer*** dans le type ***Real***.

Il est possible d'utiliser des expressions booléennes dans l'affectation. Par exemple, en ayant déclarer :

```
Var
a , b: integer;
c: boolean;
```

Il est possible d'utiliser l'affectation suivante :

```
c:=a>b;
```

## 5.3.L'affichage

Le langage Pascal utilise les instructions ***write*** et ***writeln*** pour afficher du texte ou le contenu d'une variable sur l'écran. L'instruction ***write*** permet d'afficher le texte tout en laissant le curseur sur la même ligne, alors que l'instruction ***writeln*** affiche le texte et provoque un retour à la ligne.

Syntaxe:

***Writeln(E);*** où E est une expression.

**Exemples**

Soit A une variable de type ***Integer*** dont la valeur est égale à 3.

- ***Write(A)*** affiche la valeur 3 sur l'écran.
- ***Write('A')*** affiche la lettre A sur l'écran.
- ***Write('A=', A)*** affiche le message A=3.
- ***Write(A*3)*** affiche la valeur 9.
- ***Write(A, B)*** permet d'afficher les valeurs respectives des variables A et B.

**Remarque**

Si le texte à afficher contient une apostrophe celle-ci doit être doublée. Par exemple pour afficher le message ***il fait beau aujourd'hui***, il faut exécuter l'instruction :

***write('il fait beau aujourd''hui')***

## 5.4.La lecture

L'instruction de lecture permet à l'utilisateur de transmettre des données au programme. Le langage Pascal propose deux instructions de lecture ***Read*** et ***Readln***. A la différence de l'instruction ***Read***, l'instruction ***Readln*** permet de faire un retour à la ligne après la lecture de la donnée.

Syntaxe

***Read(A);***

Avec A : nom de variable.

L'instruction ***Read(A)*** permet de récupérer la valeur saisie par l'utilisateur et la stocker dans la variable A.

On peut regrouper plusieurs ordres de lecture en un seul : ***Read(A, B, C)*** est équivalente à ***Read(A) ; Read(B) ; Read(C)***;

## 5.5.Exercice d'application

Dans cet exercice nous allons compléter le programme que nous avons créé durant l'exercice d'application de la séance précédente.

1. Ouvrir le fichier du programme "tp2.pas".
2. Dans la section exécutable (section d'instructions) ajouter les instructions nécessaires pour :
    a. Afficher le message ***Donner le rayon du cercle***.
    b. Lire la valeur du rayon et la stocker dans la variable R.
    c. Calculer le périmètre du cercle et l'affecter à la variable P.
    d. Calculer la surface du cercle et l'affecter à la variable S.
    e. Afficher la valeur du périmètre sous la forme : ***Le périmètre du cercle=valeur***
    f. Afficher la valeur de la surface sous la forme : ***La surface du cercle=valeur***
3. Compiler et exécuter le programme avec R=2.5 puis avec R=3.

## 5.6. Corrigé de l'exercice d'application

Le programme cercle créé précédemment et stocké dans le fichier tp2.pas se présente comme suit :

```
program cercle;
uses wincrt
const
pi=3.14;
var
R, P, S : real;
Begin


end.
```

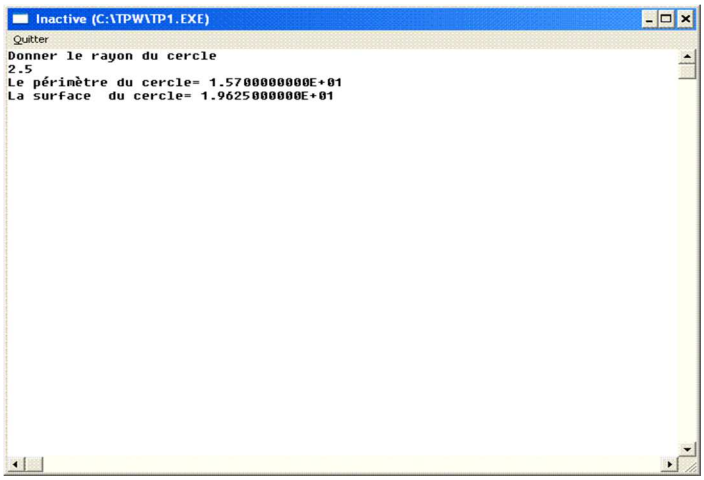
Il est demandé dans cet exercice de modifier la section exécutable (la section entre ***begin*** et ***end***.) en ajoutant des instructions.

a. Pour afficher le message ***Donner le rayon du cercle,*** il faut utiliser l'instruction ***write*** ou ***writeln*** si on veut ajouter un retour à la ligne.
   ***writeln('Donner le rayon du cercle');***
b. Pour lire la valeur du rayon et la stocker dans la variable R, il faut utiliser l'instruction Read ou Readln si on veut ajouter un retour à la ligne.
   ***Read(R);***
c. Pour Calculer le périmètre du cercle et l'affecter à la variable P, il faut utiliser l'instruction d'affectation. Le périmètre du cercle étant égale à 2*π*R, alors nous remplaçons π par la constante pi que nous avons déclaré précédemment et nous affectons le résultat à la variable P comme suit : ***P:=2*pi*R;***
d. De même, nous calculons la surface par l'instruction: ***S:=pi*R*R ;*** Il est possible d'utiliser la fonction SQR qui permet de calculer le carrée d'un nombre. L'instruction se présentera comme suit : ***S:=pi*SQR(R);***
e. Pour l'affichage du résultat, nous utilisons l'instruction writeln avec deux paramètres. Le premier correspond au message à afficher à savoir ***'Le périmètre du cercle='***, le deuxième paramètre correspond à la valeur du périmètre calculée par le programme et qui est stockée dans la variable P. L'instruction à utiliser est
   ***Writeln('Le périmètre du cercle=', P);***
f. L'affichage de la surface se fera de la même manière en utilisant l'instruction:
   ***Writeln('La surface du cercle=',S);***

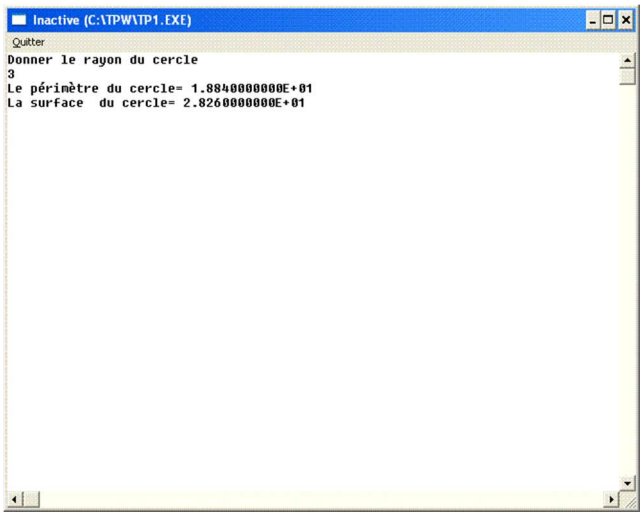
Le programme final se présente comme suit :

```
program cercle;
uses wincrt;
const
pi=3.14;
var
R, P, S : real;
Begin
 writeln('Donner le rayon du cercle');
 Read(R);
 P:=2*pi*R;
 S:=pi*R*R ;
Writeln('Le périmètre du cercle=', P);
Writeln('La surface  du cercle=',S);
end.
```

Pour exécuter le programme, il suffit de cliquer sur le menu Exécuter->Exécuter. Le programme affiche le message ***'Donner le rayon du cercle'*** et attends que l'utilisateur introduit la valeur du rayon. Nous introduisons la valeur 2.5 puis nous appuyons sur la touche Entrée du clavier. Le programme affiche le périmètre et la surface.




Nous procédons de la même manière pour exécuter le programme avec la valeur 3 du rayon.




**Remarque**

L'affichage des résultats de type ***Real*** se fait en utilisant la notation scientifique, ainsi la valeur 2.826000000E+01 est équivalente à 28.26. Pour afficher les valeurs en format ordinaire, il faut utiliser

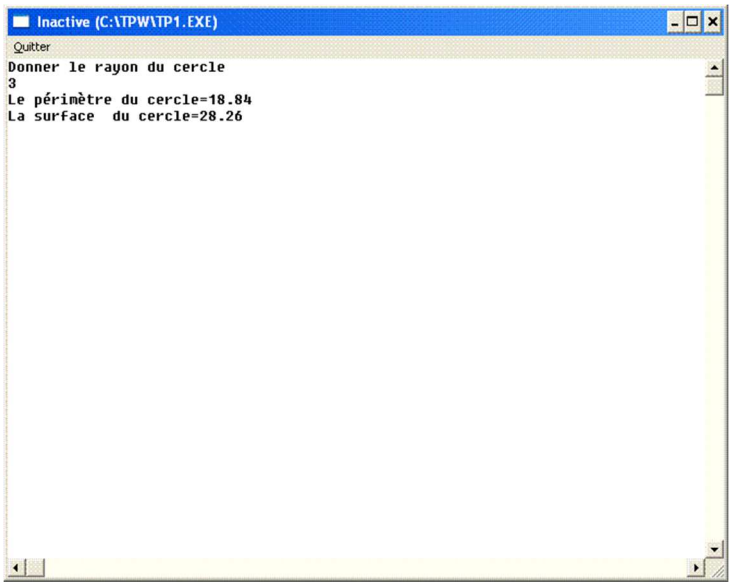
***Writeln('Le périmètre du cercle=', P:0:2);***

***Writeln('La surface du cercle=',S:0:2);***

Où

0 représente le nombre total de caractères à afficher.

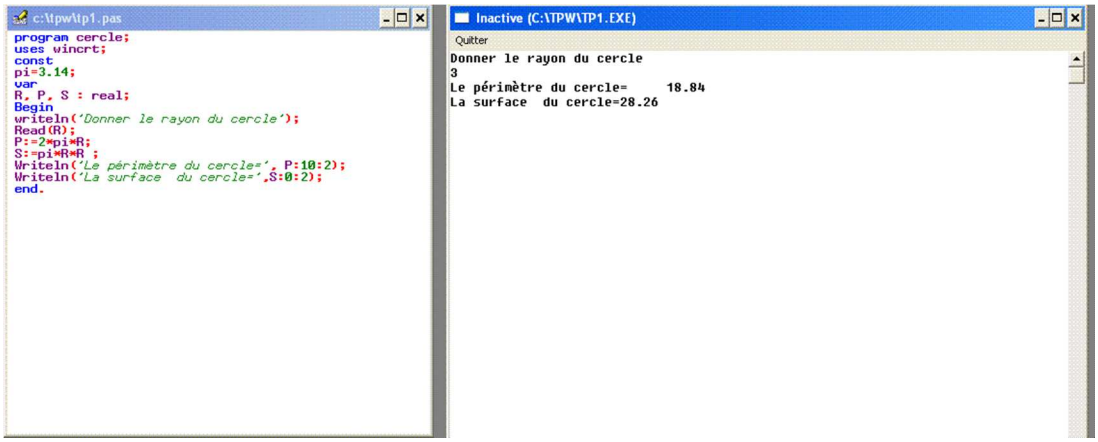
2 représente le nombre de chiffres après la virgule.




Par exemple l'instruction ***Writeln('Le périmètre du cercle=', P:10:2);*** permet d'afficher le résultat avec deux chiffres après la virgule comme suit:

***Le périmètre du cercle= 18.84.***

La longueur du nombre à afficher est 5 caractères (4 chiffres plus le point) et on demande que l'affichage soit sur 10 caractères, alors TPW ajoute 5 espaces avant le nombre.



# 6. TP N°5- Les conditions-

## 6.1.Objectifs

A la fin de cette séance, l'étudiant sera capable de:

- Exprimer des conditions simples et complexes
- Exprimer des conditions imbriquées
- Utiliser l'instruction à choix multiple (Case)

## 6.2.Exprimer des conditions simples et complexes

Pour exprimer une condition dans un programme Pascal, il faut utiliser l'instruction (***if***). Selon le cas, deux versions sont disponibles :

| **conditionnel simple** | **conditionnel alternative** |
|---|---|
| **Le conditionnel** | **L'alternative** |
| ***if condition then*** | ***if condition then*** |
| ***instruction1;*** | ***instruction1*** |
| | ***else*** |
| | ***instruction2;*** |

Dans la structure conditionnelle, l'instruction ***instruction1*** n'est exécutée que si la ***condition*** est vérifiée. Autrement rien n'est exécuté. Par contre, dans la structure alternative, si la ***condition*** n'est pas vérifiée alors c'est l'instruction ***instruction2*** qui est exécutée.

**Remarque**

L'instruction qui précède le mot clé ***else*** ne doit jamais comporter un point-virgule.

**Remarque**

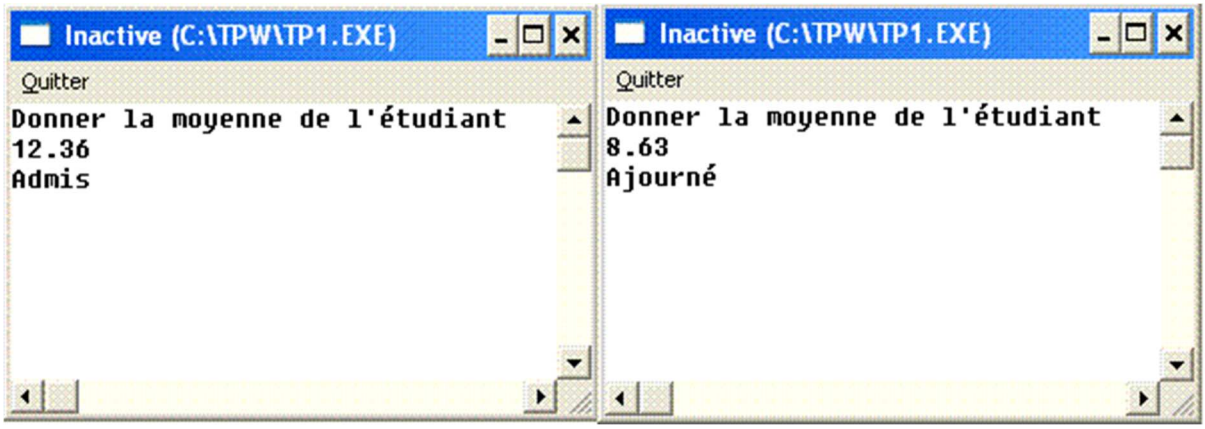
Si le traitement à exécuter dans le ***if*** ou le ***else*** est composé de plus d'une instruction, alors ces dernières doivent être délimitées par ***begin*** et ***end***; .

| **Le conditionnel** | **L'alternative** |
|---|---|
| ***if condition then*** | ***if condition then*** |
| ***begin*** | ***begin*** |
| ***inst1;*** | ***inst1;*** |
| ***inst2;*** | ***inst2;*** |
| ***end***; | ***end*** |
| | ***else*** |
| | ***begin*** |
| | ***inst3;*** |
| | ***inst4;*** |
| | ***end;*** |

## 6.3.Exemple illustratif 1

Le programme suivant permet de lire la moyenne d'un étudiant et affiche la mention "Admis" si la moyenne est supérieure ou égale 10, et la mention "Ajourné" dans le cas contraire.

```
program mention;
uses wincrt;
var
m :real;
Begin
Writeln('Donner la moyenne de l''étudiant');
read(m);
if  m>=10 then
    writeln('Admis')
   else
   writeln('Ajourné');
end.
```




Il est possible combiner des conditions simples pour former une condition complexe. La combinaison se fait à l'aide des opérateurs logiques (***not, and, or*** et ***xor***).

**Exemple**

```
if (a>0) and (b>0) then
   inst1;
```

Dans cet exemple, l'instruction ***inst1*** n'est exécutée que si les deux variables ***a*** et ***b*** à la fois sont nulles.

```
if (a>0) or (b>0) then
   inst1;
```

Par contre, dans le deuxième exemple, l'instruction ***inst1*** est exécutée si au moins l'une des deux variables est nulle.

## 6.4.Exprimer des conditions imbriquées

Une structure ***If-then-else*** ne permet de traiter que deux traitements mutuellement exclusif. Cependant, quand le nombre de cas à traiter dépasse deux, une seule instruction ***If-then-else*** ne permet pas de résoudre le problème. A cet effet, il est nécessaire d'imbriquer des instructions ***If-then-else*** pour pouvoir traiter tous les cas possibles.

**Exemple:**

```
if   cond1   then
     if  cond2    then
        inst1
        else
           inst2
              else
              if  cond3   then
                 inst3
                 else
                    inst4 ;
```

Dans cet exemple, nous avons quatre traitements à exécuter suivant trois différentes conditions.

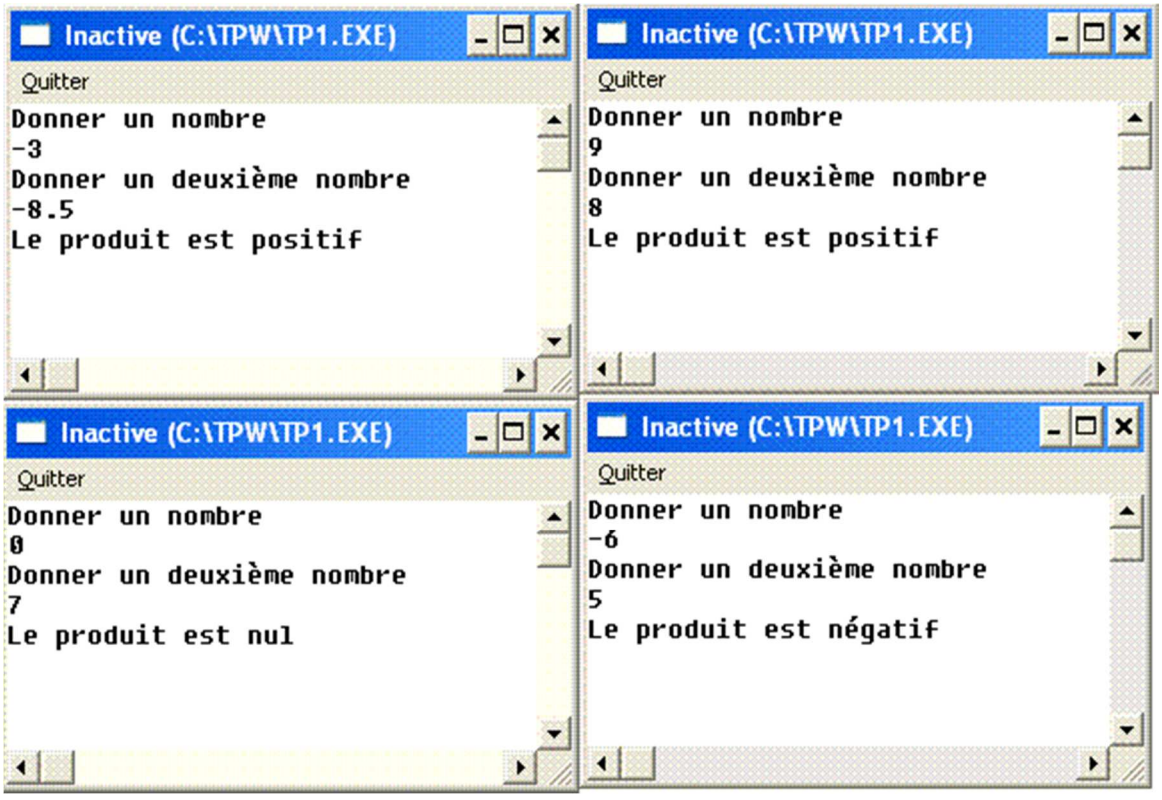
- L'instruction ***inst1*** est exécutée quand les deux conditions ***cond1*** et ***cond2*** sont vérifiées.
- L'instruction ***inst2*** est exécutée quand la condition ***cond1*** est vérifiée et la condition ***cond2*** n'est pas vérifiée.
- L'instruction ***inst3*** est exécutée quand la condition ***cond3*** est vérifiée et la condition ***cond1*** n'est pas vérifiée.
- L'instruction ***inst4*** est exécutée quand les deux conditions ***cond1*** et ***cond3*** ne sont pas vérifiées.

## 6.5.Exemple illustratif 2

Le programme suivant permet d'afficher le signe du produit de deux nombres sans calculer le produit. On distingue 3 cas possibles :

- Le produit est positif si les deux nombres ont le même singe (les deux sont positifs ou les deux sont négatifs).
- Le produit est nul si l'un des deux nombres est nul.
- Le produit est négatif s'il est ni nul ni positif.

```
program signe;
uses wincrt;
var
a, b :real;
Begin
Writeln('Donner un nombre ');
read(a);
Writeln('Donner un deuxième nombre');
read(b);
if ((a>0) and (b>0)) or ((a<0) and (b<0)) then
    writeln('Le produit est positif')
    else
       if(a=0) or (b=0) then
          writeln('Le produit est nul')
           else
          writeln('Le produit est négatif');
end.
```



## 6.6.L'instruction à choix multiple (case)

L'instruction à choix multiple noté « **case** » est une instruction qui permet d'effectuer un branchement à partir de la valeur d'une variable. Elle peut dans certains cas remplacer une série de ***if-then-else*** imbriqués. On l'utilise lorsque les choix à gérer sont multiples.

Dans l'instruction « **case »**, la condition de branchement doit porter sur la comparaison d'une variable avec des valeurs constantes, et non sur une condition quelconque (le branchement en fonction de conditions comme delta>0, delta<0 ou delta=0 doit être réalisés avec ***if-then-else*** imbriquées)

L'instruction « **case »** évalue l'expression qui lui est donnée et branche le programme vers le cas correspondant à la valeur effective de cette expression.

**Syntaxe**

**case expression of**
**cas_1 : instruction_1;**
**cas_2 : instruction_2;**
**.....**
**cas_n : instruction_n ;**
**else : autre_instruction ;**
**end ;**

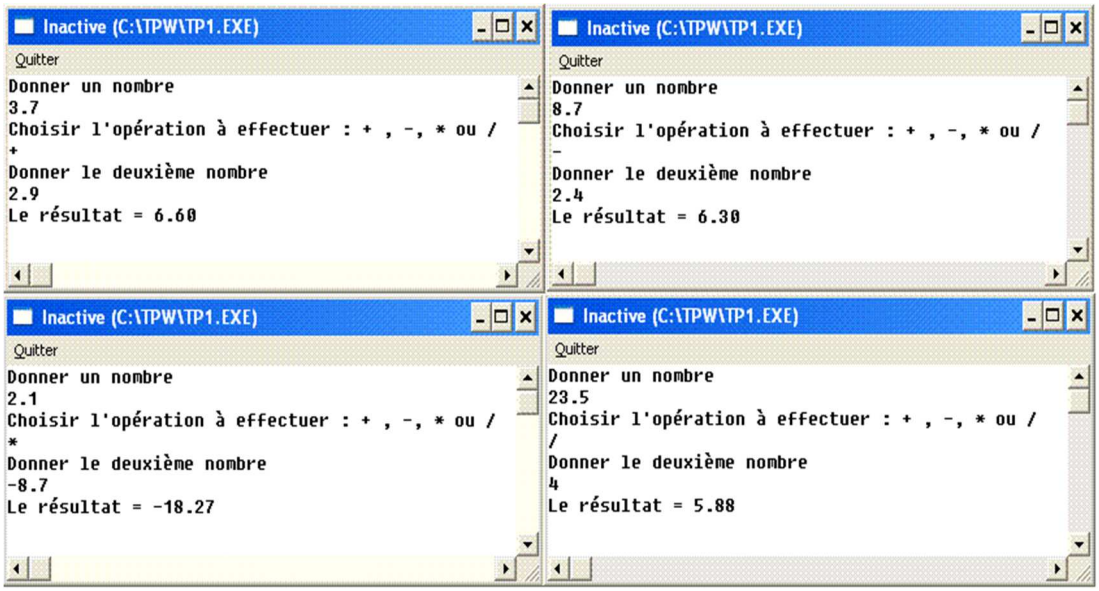
## 6.7.Exemple illustratif 3

Le programme suivant permet de simuler une calculatrice à quatre opérations. L'utilisateur introduit d'abord deux nombres. Puis, il introduit le singe de l'opération à effectuer (+, -, * ou /). Pour réaliser ce programme, on a besoin de :

- Trois variables (a, b , res) pour représenter, respectivement, les nombres introduits par l'utilisateur et le résultat de l'opération.
- Une variable (op) de type caractère pour représenter l'opération à effectuer.

```
Program calculatrice ;
Uses wincrt ;
Var
a, b, res :real ;
op : char ;
begin
Writeln('Donner un nombre') ;
Readln(a) ;
Writeln('Choisir l''opération à effectuer : + , -, * ou /') ;
Readln(op) ;
```

```
Writeln('Donner le deuxième nombre') ;
Readln(b) ;
Case op  of
'+' : res :=a+b ;
'-' : res :=a-b ;
'*' : res :=a*b ;
'/' : res :=a/b ;
End ;
Writeln('Le résultat = ', res :0 :2) ;
End.
```




## 6.8.Exercice d'application

Ecrire un programme qui lit la moyenne d'un étudiant puis affiche l'une des mentions suivantes :

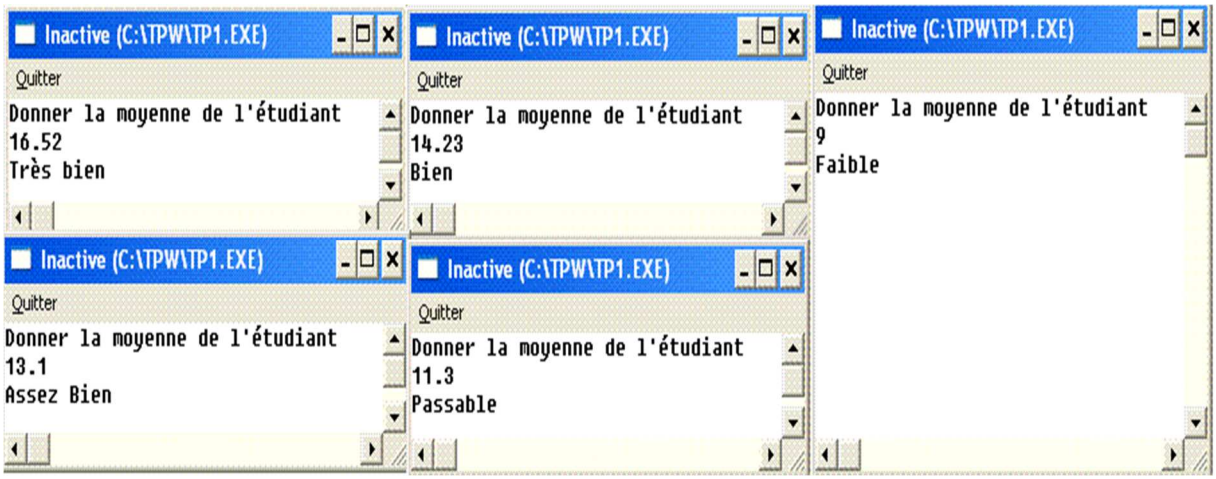
- 'Très bien' : si la moyenne de l'étudiant est supérieure ou égale à 16.
- 'Bien' : si la moyenne de l'étudiant est comprise entre 14 et 16 (14<=moyenne<16).
- 'Assez Bien' : si  la moyenne est comprise entre 12 et 14 (12<=moyenne<14).
- 'Passable' si la moyenne est comprise entre 10 et 12 (10<=moyenne<12).
- 'Faible' si la moyenne est inférieure à 10.

## 6.9.Corrigé de l'exercice d'application

Pour résoudre ce problème, nous avons besoin d'utiliser une variable m pour stocker la moyenne de l'étudiant.

Pour traiter les 5 cas possibles, if faut imbriquer 4 instructions if-then-else. L'utilisation du Case dans cet exercice n'est pas possible car les conditions sont exprimée sous forme d'inégalités (>=, <…). Le programme se présente comme suit :

```
program mention;
uses wincrt;
var
m :real;
Begin
Writeln('Donner la moyenne de l''étudiant');
read(m);
if  m>=16 then
    writeln('Très bien')
    else
    if m>=14 then
     writeln('Bien')
    else
     if m>=12 then
    writeln('Assez Bien')
    else
    if m>=10 then
    writeln('Passable')
     else
    Writeln('Faible') ;
end.
```



# 7. TP N°6- Les boucles-

## 7.1.Objectifs

A la fin de cette séance, l'étudiant sera capable d'utiliser :

- La boucle For
- La boucle Repeat
- La boucle While

## 7.2.La boucle For

Cette structure exprime la répétition d'un traitement un nombre de fois. Dans cette boucle le nombre de répétitions est connu. Le nombre de répétitions est déterminé à l'aide de deux valeurs (***v_initiale*** et ***v_finale***). Le traitement est donc exécuté une fois pour chaque valeur de la variable ***var*** de ***v_initiale*** jusqu'à ***v_finale***.

**Syntaxe**

***for var := v_initiale to v_finale do***

***traitement***

ou

***for var := v_initiale to v_finale do***

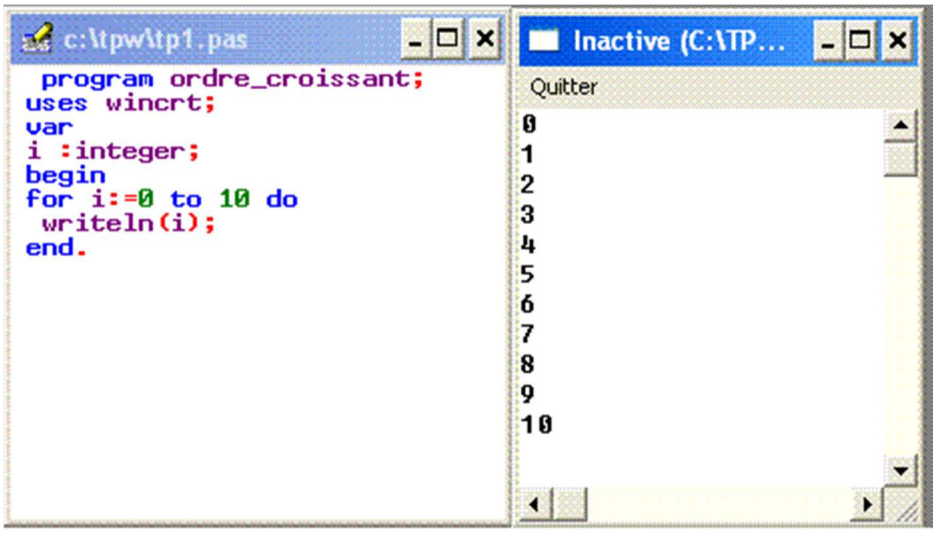
***traitement***

La variable ***var*** (nommée aussi compteur de la boucle) et les valeurs ***v_initiale*** et ***v_finale*** doivent être de type ***integer***.

Avec ***to***, la valeur de la variable de contrôle est incrémentée de 1 à chaque tour de boucle. Avec ***downto***, la valeur de la variable de contrôle est décrémentée de 1 à chaque tour de boucle.

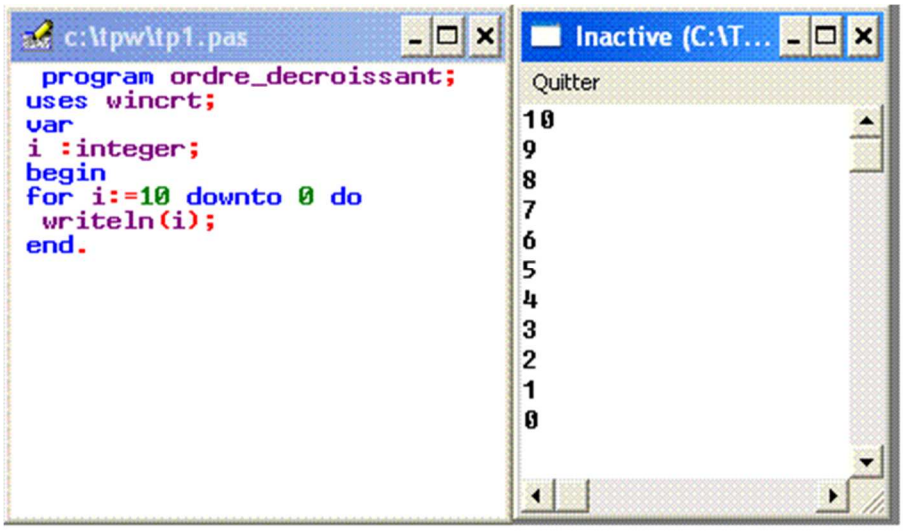
**Exemple1**

Le programme suivant permet d'afficher les nombres de 0 à 10 par ordre croissant.



**Exemple 2**

Le programme ci-dessous affiche les nombres de 0 à 10 mais dans l'ordre inverse.




**Remarque**

Si le traitement à exécuter dans la boucle ***for*** est composé de plus d'une instruction, alors il doit être délimité par les mots clés ***begin*** et ***end ;***

## 7.3.La boucle Repeat

Cette boucle permet de répéter l'exécution d'un traitement jusqu'à ce qu'une condition d'arrêt soit vérifiée.

**Syntaxe**

***repeat***

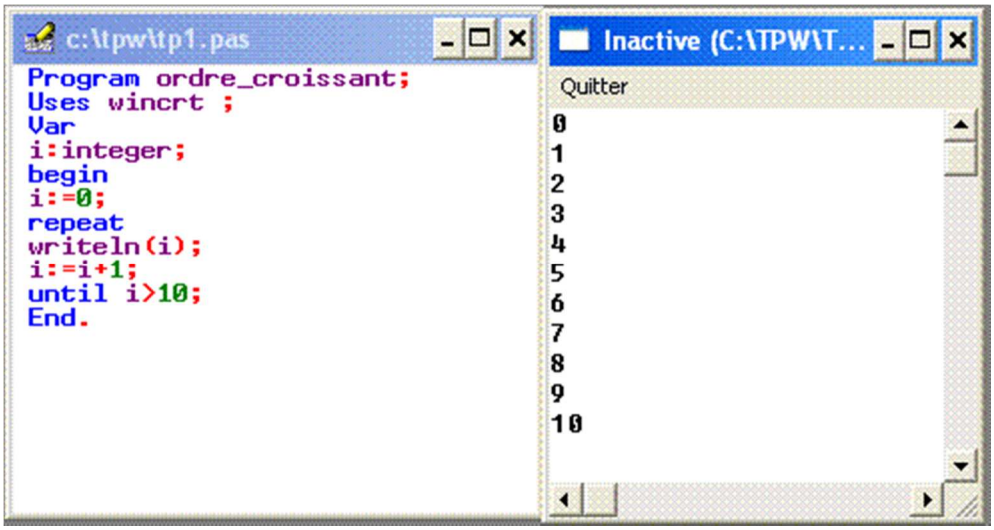
***traitement***

***until condition ;***

Le traitement est exécuté au moins une fois même si la condition d'arrêt est vérifiée dès le début de l'exécution.

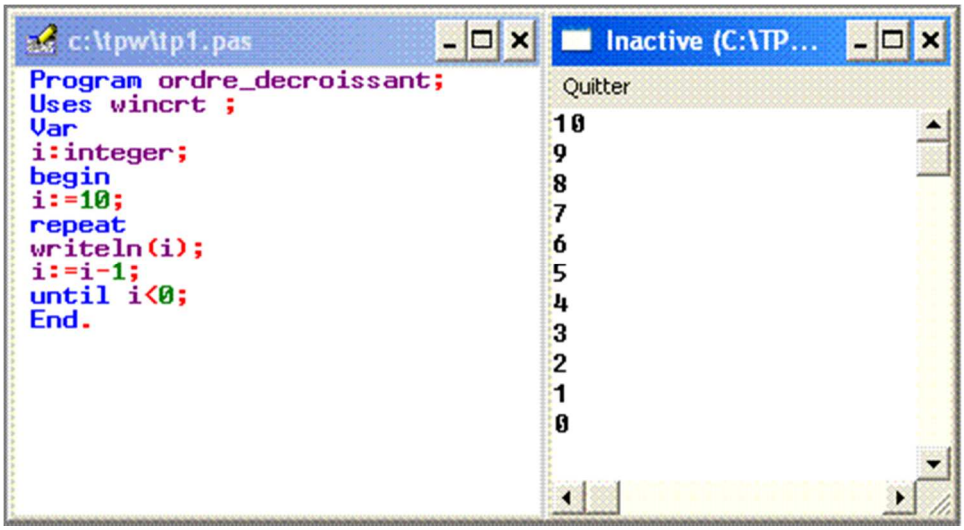
**Exemple**

Nous reprenons l'exemple qui permet d'afficher les nombres de 0 à 10 dans l'ordre mais cette fois-ci avec la boucle **Repeat**.




Pour avoir un affichage par ordre décroissant, il suffit de :

- Initialiser le compteur i à 10.
- Décrémenter le compteur par 1 à chaque exécution.
- Sortir de la boucle quand i<0.




## 7.4.La boucle While

Cette structure permet de répéter l'exécution d'un traitement **zéro** ou plusieurs fois et de s'arrêter lorsque la condition d'exécution n'est plus vérifiée. En effet, lorsque la condition d'exécution est vérifiée, le traitement est exécuté, sinon la boucle s'arrête.

**Syntaxe**

***While condition do***

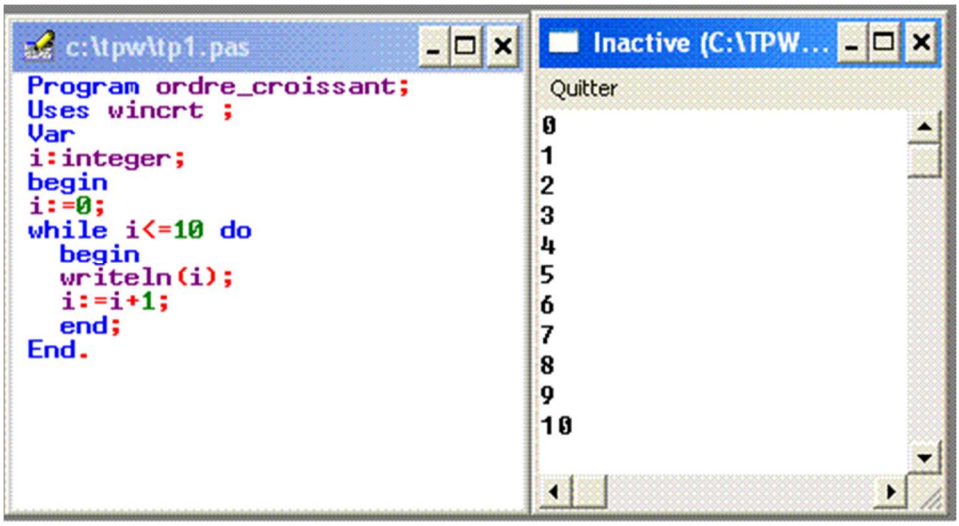
***Traitement***

**Remarque**

Si le traitement à exécuter dans la boucle ***while*** est composé de plus d'une instruction, alors il doit être délimité par les mots clés ***begin*** et ***end ;***

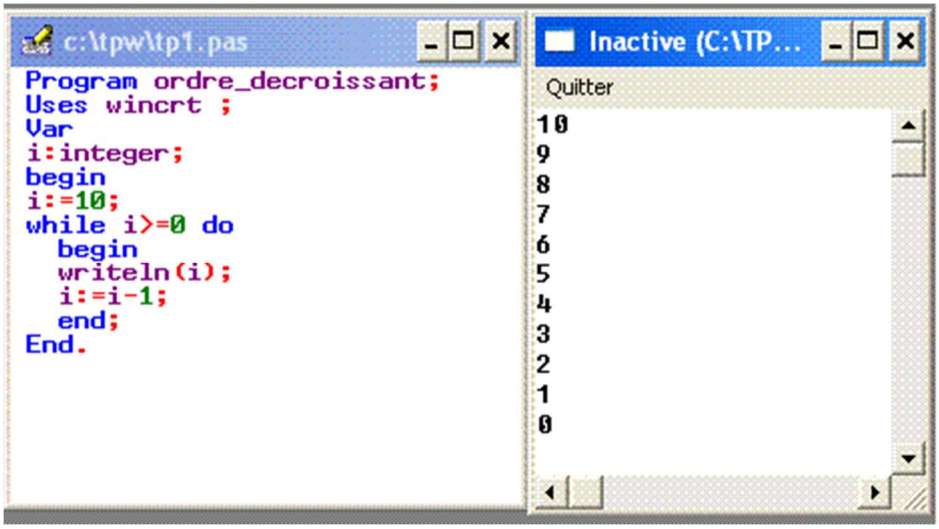
**Exemple**

Nous reprenons l'exemple qui permet d'afficher les nombres de 0 à 10 dans l'ordre mais cette fois-ci avec la boucle ***while***.




Pour avoir un affichage par ordre décroissant, il suffit de :

- Initialiser le compteur i à 10.
- Répéter l'exécution du traitement tant que i>=0
- Décrémenter le compteur par 1 à chaque exécution.




## 7.5.Exercice d'application 1

Ecrire un programme qui calcule et affiche la moyenne de n nombres introduits au clavier.

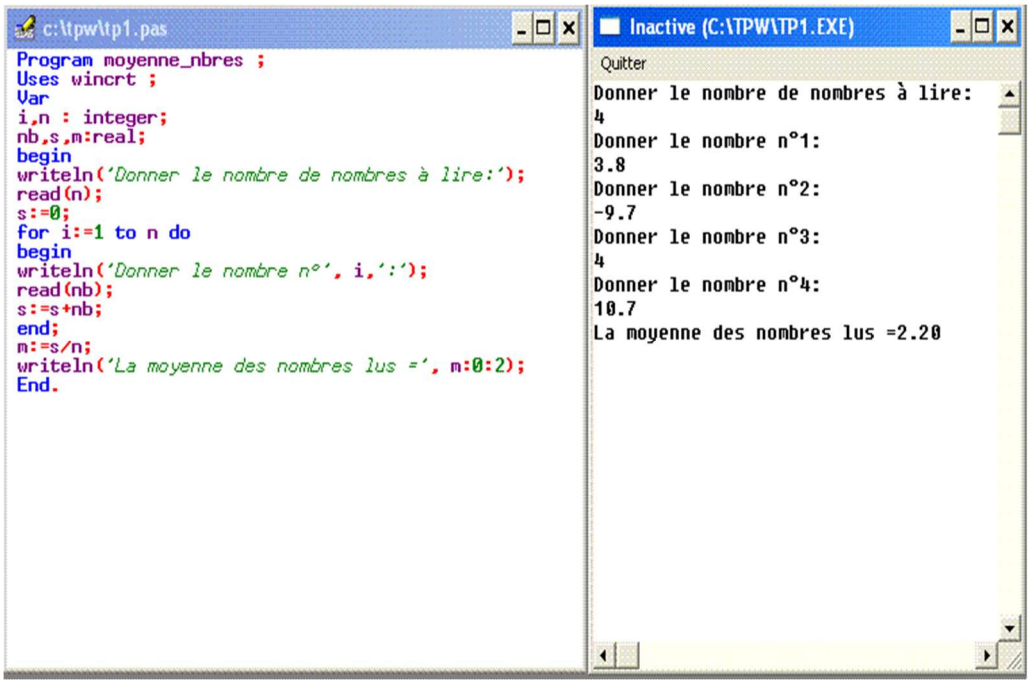
## 7.6.Corrigé de l'exercice d'application

Dans ce programme, on doit lire n nombres, calculer leur somme et leur moyenne. L'opération de lecture doit se répétée n fois. Après chaque lecture, le programme doit ajouter la valeur du nombre lu à la somme des nombres lus précédemment. Donc le traitement qui consiste à lire le nombre et à l'ajouter à la somme doit être répété n fois.

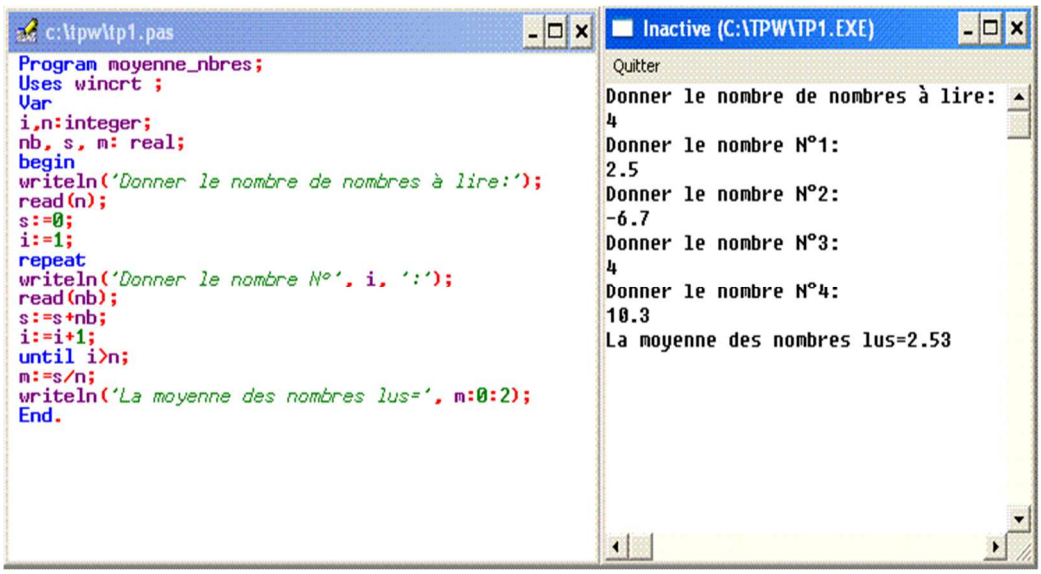
Pour résoudre ce problème, nous avons besoin de déclarer les variables suivantes :

- **n** de type ***integer*** qui représente le nombre de nombres à lire.
- **nb** de type ***real*** qui représente les nombres à lire.
- **S** de type ***real*** qui représente la somme des nombres lus.
- **M** de type ***real*** qui représente la moyenne des nombre lus.
- **i** de type ***integer*** qui représente le compteur de la boucle.

***Avec la boucle for***



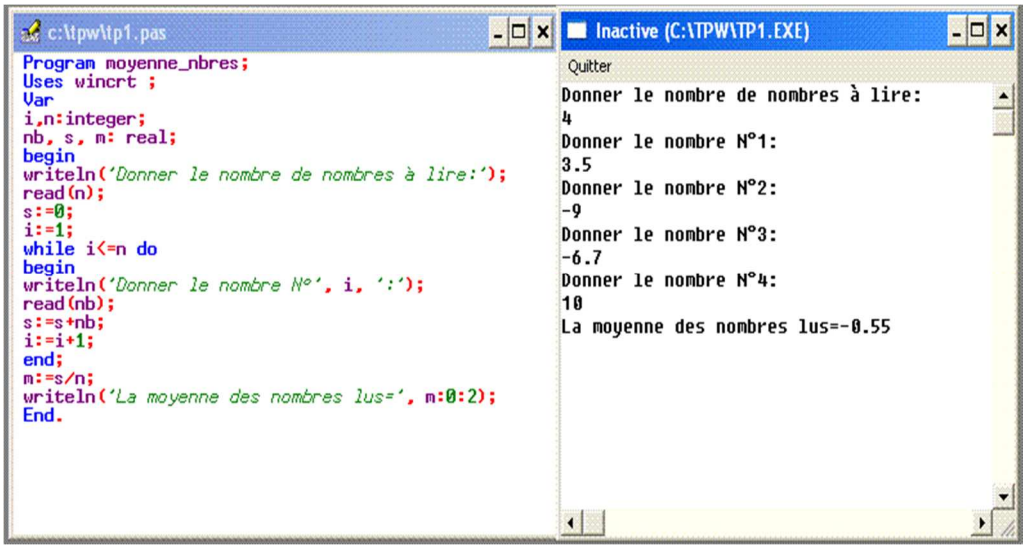
***Avec la boucle Repeat***



```
Program moyenne_nbres;
Uses wincrt ;
Var
i,n:integer;
nb, s, m: real;
begin
writeln('Donner le nombre de nombres à lire:');
read(n);
s:=0;
i:=1;
repeat
writeln('Donner le nombre N°', i, ':');
read(nb);
s:=s+nb;
i:=i+1;
until i>n;
m:=s/n;
writeln('La moyenne des nombres lus=', m:0:2);
End.
```

```
Donner le nombre de nombres à lire:
4
Donner le nombre N°1:
2.5
Donner le nombre N°2:
-6.7
Donner le nombre N°3:
4
Donner le nombre N°4:
10.3
La moyenne des nombres lus=2.53
```

***Avec la boucle While***



```
Program moyenne_nbres;
Uses wincrt ;
Var
i,n:integer;
nb, s, m: real;
begin
writeln('Donner le nombre de nombres à lire:');
read(n);
s:=0;
i:=1;
while i<=n do
begin
writeln('Donner le nombre N°', i, ':');
read(nb);
s:=s+nb;
i:=i+1;
end;
m:=s/n;
writeln('La moyenne des nombres lus=', m:0:2);
End.
```

```
Donner le nombre de nombres à lire:
4
Donner le nombre N°1:
3.5
Donner le nombre N°2:
-9
Donner le nombre N°3:
-6.7
Donner le nombre N°4:
10
La moyenne des nombres lus=-0.55
```

## 7.7.Exercice d'application 2

Ecrire un programme lit une suite de nombres. La lecture se termine par la saisie du 0. Le programme affiche ensuite le nombre de nombres lus (le 0 n'est pas comptabilisé).

## 7.8.Corrigé de l'exercice d'application 2

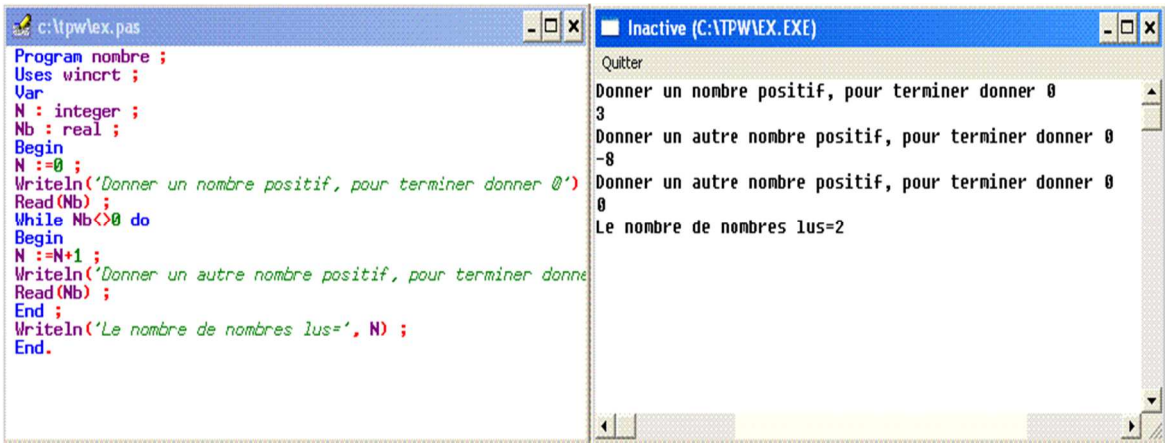
A la différence des exercices précédent, dans cet exercice le nombre de nombres à lire n'est pas connu à l'avance (c'est au programme de le calculer). Donc, le nombre de répétition n'est pas connu et par conséquent il n'est pas possible d'utiliser la boucle ***for***. Nous devons donc écrire notre programme soit par la boucle ***Repeat*** soit par la boucle ***While***. La boucle s'arrête quand le nombre lu est égal à 0.

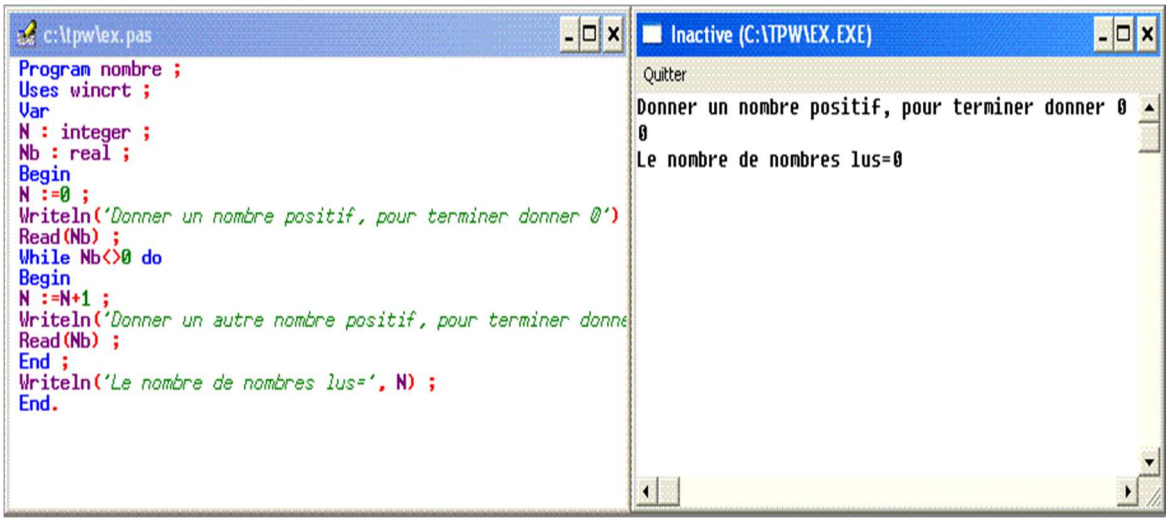
Pour résoudre ce problème, nous avons besoin de déclarer les variables suivantes :

- **N** de type ***integer*** qui représente le nombre de nombres lus.
- **Nb** de type ***real*** qui représente les nombres à lire.

***Avec la boucle while***

D'abord, il faut initialiser la variable N à zéro avant la boucle. La boucle ***while*** s'exécute quand le nombre lu est différent de 0. Donc, il faut faire une première lecture pour avant la boucle.
A chaque exécution de la boucle, le nombre de nombres lu (N) est incrémenté de 1 à fin de compter le nombre de nombres lus.

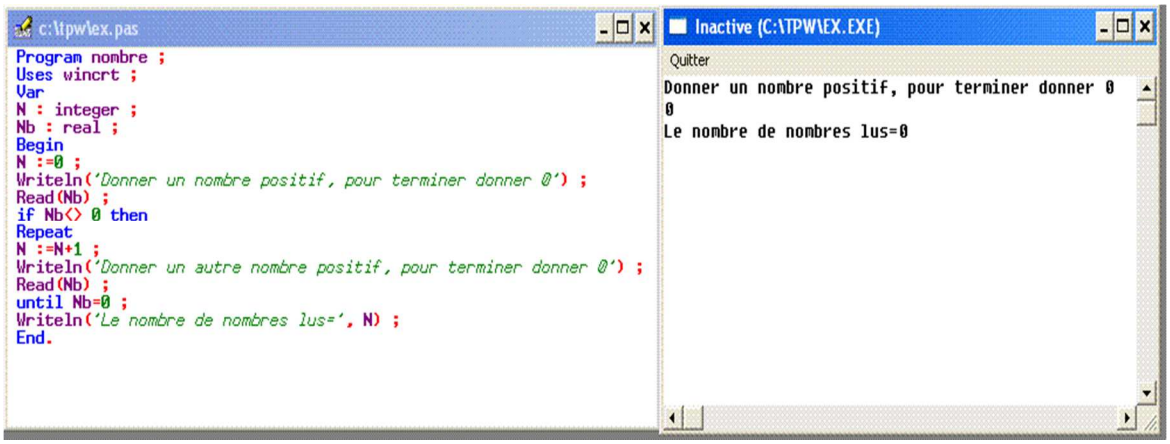


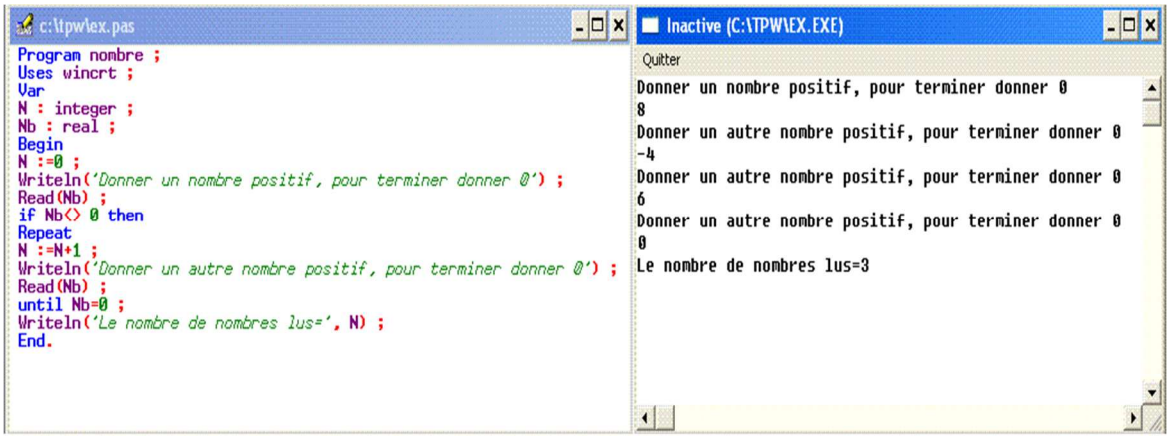




***Avec la boucle Repeat***

Dans la boucle ***Repeat***, les répétitions s'arrêtent quand le nombre lu est égal à zéro. Dans ce type de boucle, l'évaluation de la condition se fait à la fin de la boucle. Ce comportement risque de poser un problème si l'utilisateur commence par introduire le 0 (ce cas ne pose aucun problème pour la boucle while puisque la condition est évaluée avant de rentrer dans la boucle).

Pour éviter ce problème, nous devons ajouter une condition avant la l'exécution de la boucle afin de s'assurer que le premier nombre lu est différent de 0. En d'autres termes, cette condition permet d'exécuter la boucle ***Repeat*** que si le premier nombre lu est différent de zéro.



# 8. TP N°7-Les Tableaux à une dimension (Les Vecteurs)-

## 8.1.Objectifs

A la fin de cette séance, l'étudiant sera capable de :

- Déclarer un tableau à une dimension.
- Remplir un tableau à une dimension.
- Afficher le contenu d'un tableau à une dimension.

## 8.2.Déclarer un tableau à une dimension

Trois informations sont nécessaires pour déclarer un tableau à une dimension :

- Le nom du tableau.
- La taille du tableau (le nombre d'éléments dans le tableau).
- Le type de ses éléments.

**Syntaxe :**

**Nom_Tab : Array [1..taille] of type ;**

**Exemples**

**Tab :array [1..10] of integer :** Ce code permet de déclarer un tableau nommé **Tab** de 10 éléments de type entier.

**Tab :array [1..5] of char :** Ce code permet de déclarer un tableau nommé **Tab** de 5 éléments de type caractère.

## 8.3.Remplir un tableau à une dimension

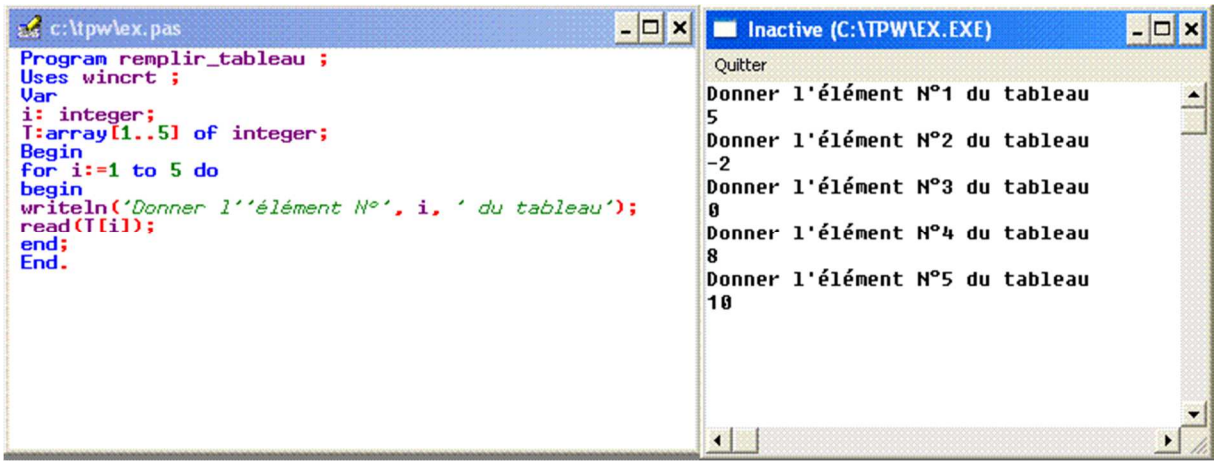
Pour remplir la $i^{ème}$ case d'un tableau ***vect*** par une valeur saisie par l'utilisateur, il faut utiliser l'instruction ***Read*** comme suit :

**Read(vect[i])**

Pour remplir tous les éléments d'un tableau, il est nécessaire d'utiliser une boucle. Puisque la capacité du tableau est connue, alors la boucle ***« for »*** est la plus appropriée pour ce traitement mais rien n'interdit d'utiliser les autres boucles.

**Exemple**

Le programme ci-dessous permet de déclarer et remplir un tableau **T** de 5 éléments de type ***integer***.




## 8.4.Afficher le contenu d'un tableau à une dimension

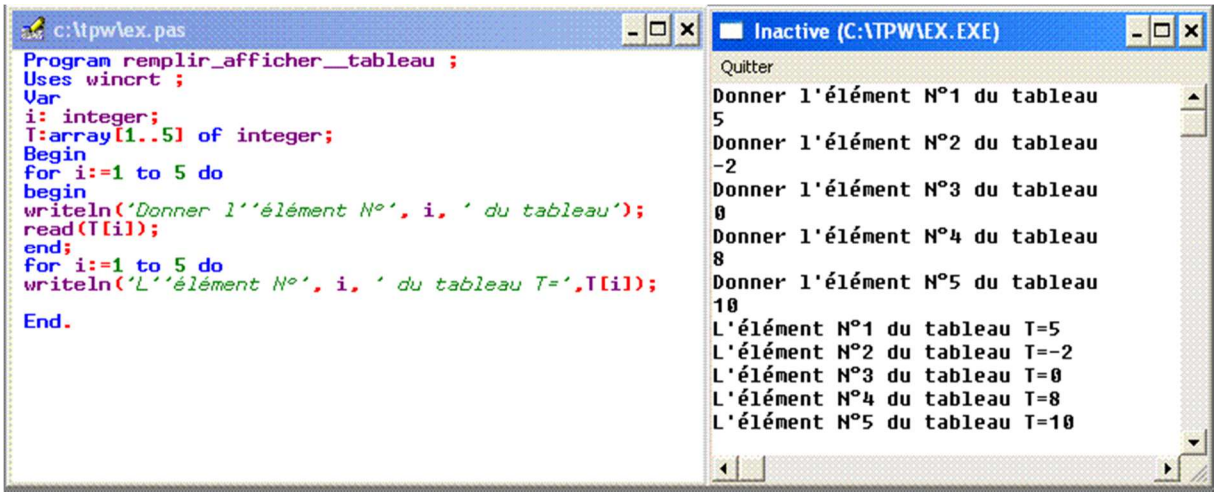
Pour afficher la ***i*ème** case d'un tableau ***vect***, il faut utiliser l'instruction ***writeln*** comme suit :

**writeln( vect[i])**

Pour afficher tous les éléments d'un tableau, il est nécessaire d'utiliser une boucle. Puisque la taille du tableau est connue, alors la boucle ***« for »*** est la plus appropriée pour ce traitement mais rien n'interdit d'utiliser les autres boucles.

**Exemple**

Le programme suivant affiche les éléments du tableau rempli précédemment.




## 8.5.Exercice d'application

Ecrire un programme qui permet de remplir de lire 5 mots et de les stocker dans un tableau. Le programme affiche, ensuite, le mot le plus long ainsi que sa longueur.

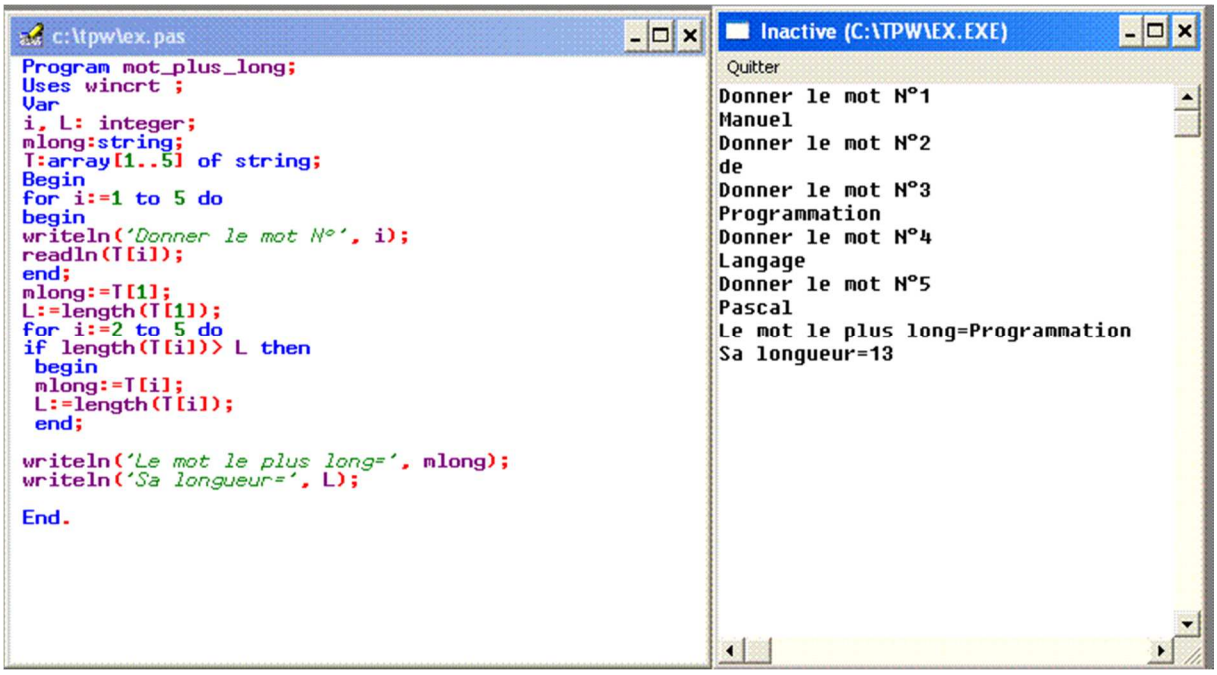
## 8.6.Corrigé de l'exercice d'application

Pour résoudre ce problème, nous avons besoin de déclarer :

- Un tableau **T** de taille 5 et type ***string*** (les mots sont des chaînes de caractère).
- Un compteur **i** de type ***integer*** pour parcourir le tableau.
- Une variable **mlong** de type ***string*** qui représente le mot le plus long.
- Une variable **L** de type integer qui représente la longueur du mot le plus long.

Une fois le tableau rempli, le programme doit le parcourir afin de rechercher le mot le plus long. Pour ce faire, on suppose que le mot le plus long se trouve dans le premier élément du tableau et on calcule sa longueur. Puis, pour élément du tableau, on vérifie si sa longueur est supérieure à celle du mot supposé. Si c'est le cas alors l'élément en cours est considéré comme le mot le plus long et sa longueur est calculée de nouveau.

Pour calculer la longueur du mot on utilise la fonction ***length()***.



# 9. TP N°8-Les Tableaux à deux dimensions (Les Matrices)-

## 9.1.Objectifs

A la fin de cette séance, l'étudiant sera capable de :

- Déclarer un tableau à deux dimensions.
- Remplir un tableau à deux dimensions.
- Afficher le contenu d'un tableau deux dimensions.

## 9.2.Déclarer un tableau à deux dimensions

Trois informations sont nécessaires pour déclarer un tableau à une dimension :

- Le nom du tableau.
- La taille du tableau en termes de nombre de lignes et de nombre de colonnes (le nombre d'éléments dans le tableau).
- Le type de ses éléments.

**Syntaxe :**

**Nom_Tab : Array [1..nombre_lignes, 1..nombre_colonnes] of type ;**

**Exemples**

**Mat :array [1..3,1..4] of integer :** Ce code permet de déclarer une matrice **Mat** de 3 lignes et 4 colonnes de type entier.

**Mat :array [1..2,1..3] of char :** Ce code permet de déclarer une matrice **Mat** de deux lignes et 3 colonnes dont les éléments sont de type caractère.

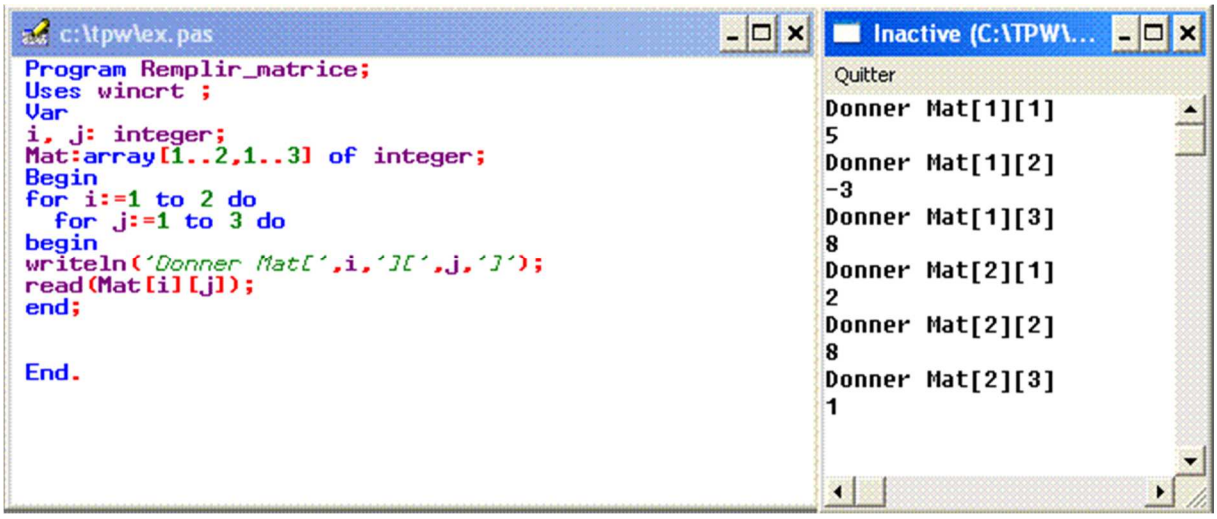
## 9.3.Remplir un tableau à deux dimensions

Pour remplir l'élément situé dans la $i^{ème}$ ligne et la $j^{ème}$ colonne d'une matrice ***« Mat »*** par une valeur saisie par l'utilisateur, on utilise l'instruction lire comme suit : ***Read(Mat[i][j]) ;***

Pour remplir une matrice, nous avons besoins deux boucles. La première permet de parcourir les lignes et la deuxième est associée aux colonnes.

### Exemple

Le programme ci-dessous permet de déclarer et remplir une matrice **Mat** de 2 lignes et 3 colonnes de type ***integer***.

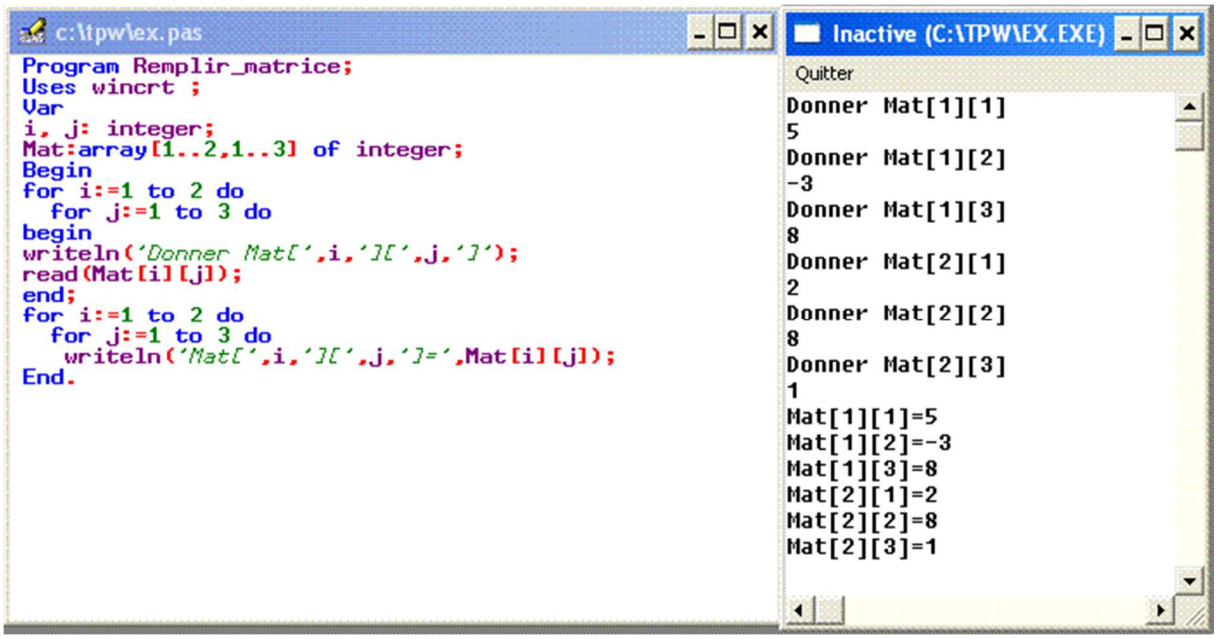



## 9.4.Afficher le contenu d'un tableau à deux dimensions

Pour afficher l'élément situé dans la ***i***ème ligne et la ***j***ème colonne d'une matrice ***« Mat »,*** on utilise l'instruction ***Writeln*** comme suit : ***Writeln (Mat[i][j]) ;***

**Exemple**

Le programme suivant affiche les éléments de la matrice remplie précédemment.




## 9.5.Exercice d'application

Ecrire un programme qui permet de remplir une matrice carrée de capacité 3 et de type integer. Le programme affiche la somme des éléments de la diagonale.

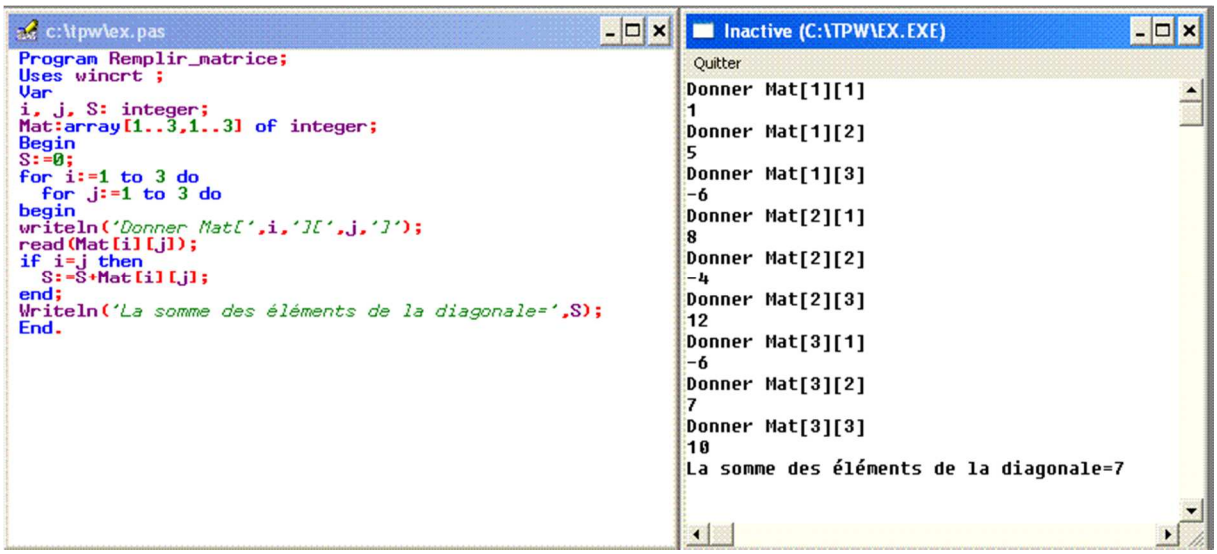
## 9.6.Corrigé de l'exercice d'application

Pour résoudre ce problème, nous avons besoin de déclarer :

- Une matrice **Mat** de 3 linges et 3 colonnes de type ***integer***.
- Deux indices (**i** et **j**) qui permettent de parcourir les lignes et les colonnes de la matrice.
- Une variable **S** de type ***integer*** qui représente la somme des éléments de la diagonale de la matrice.

Pour repérer les éléments de la diagonale, on doit vérifier que le numéro de la ligne soit égal au numéro de la colonne (i=j)

Si on vérifie sur la fenêtre d'exécution ci-dessous. Les éléments de la diagonale sont Mat[1][1], Mat[2][2], Mat[3][3]. La somme S=1-4+10=7.



# 10. TP N°9-Les enregistrements-

## 10.1. Objectifs

A la fin de cette séance, l'étudiant sera capable de :

- Déclarer un enregistrement.
- Accéder aux champs d'un enregistrement.
- Parcourir les enregistrements à l'aide de la structure ***With-do***

## 10.2. Déclarer un enregistrement

A la différence des tableaux, les enregistrements sont des structures de données dont les éléments peuvent être de type différent.

L'enregistrement représente à une structure de données composée d'éléments de type prédéfinis.

En Pascal, il existe deux façons pour déclarer un enregistrement. Nous utilisons la plus simple d'entre elles.

**Syntaxe**

**Nom_enreg : Record**

**Champs_1 : type ;**

**Champs_2 : type ;**

**....**

**Champs_n : type ;**

**End ;**

**Exemple :**

Le code suivant permet de déclarer une variable nommée **Etd** de type ***record*** qui représente un étudiant défini par son nom, son prénom, son âge et sa moyenne.

***Var***

***Etd : record***

***Nom : string ;***

***Prenom : string ;***

***Age : integer ;***

***Moyenne : real ;***

***End ;***

## 10.3. Accès aux champs d'un enregistrement

Le champ d'un enregistrement est accessible à travers son nom à l'aide de l'opérateur '.'. Ainsi, **Etd.nom** indique la valeur stockée dans le champ **nom** de la variable **Etd**.

Les champs d'un enregistrement sont considérés comme des variables et de ce fait ils peuvent subir les mêmes opérations telles que l'affectation, la lecture et l'affichage. Ainsi, pour affecter la valeur 'Rabah' au champ nom de l'enregistrement **etd**, on utilise : **Etd.nom** :=**'Rabah' ;**

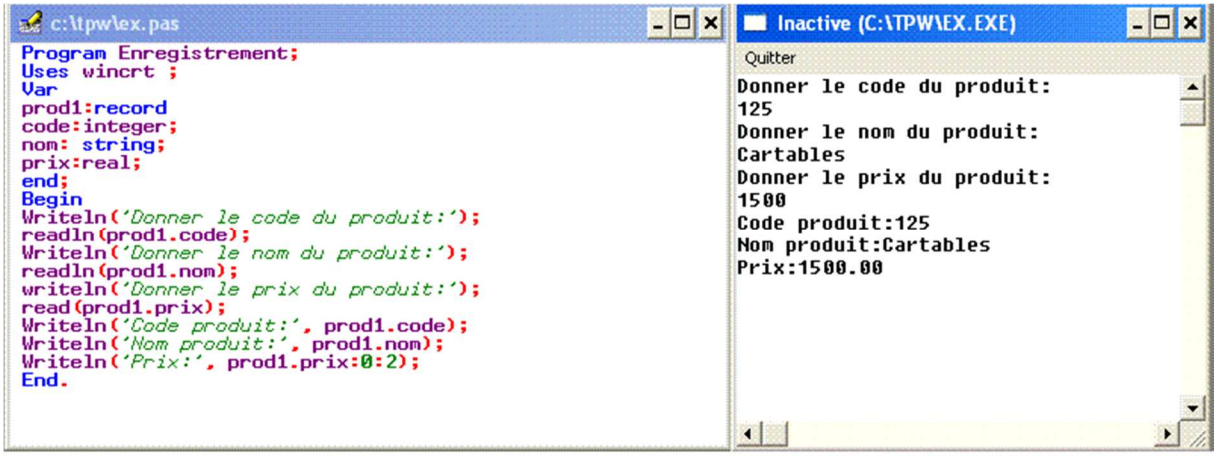
Pour remplir le champ **prenom** de l'enregistrement **etd** par une valeur saisie par l'utilisateur, on utilise : **Readln(Etd.prenom) ;**

Pour afficher la valeur stockée dans le champ **age** de l'enregistrement **etd** sur l'écran, on utilise : **Writeln (Etd.age) ;**

## 10.4. Exercice d'application

Ecrire un programme qui permet de :

1. Déclarer une variable de type record nommé prod1 représentant un produit, composé des champs suivants : « Code » de type ***integer***, « Nom » de type ***string***, « Prix » de type ***real***.
2. Remplir les champs de la variable prod1 par des valeurs saisies par l'utilisateur.
3. Afficher les champs de la variable prod1 sur l'écran.




## 10.5. Un enregistrement comme champs dans un autre enregistrement

Il est possible que le champ d'un ***record*** soit aussi de type ***record***. Par exemple, si nous voulons ajouter à la variable **Etd** déclarée précédemment, un champ représentant l'adresse de l'étudiant. Ce champ est composé du numéro de la maison, la rue et la ville. Donc le champ adresse peut être considéré comme ***record***.

Le code ci-dessous représente la nouvelle déclaration de la variable **Etd** en ajoutant le champ **adresse**.

***Var***

***Etd : record***

***Nom : string ;***

***Prenom : string ;***

***Age : integer ;***

***Moyenne : real ;***

***Adresse : record***

***Num :integer ;***

***Rue : string ;***

***Ville : string ;***

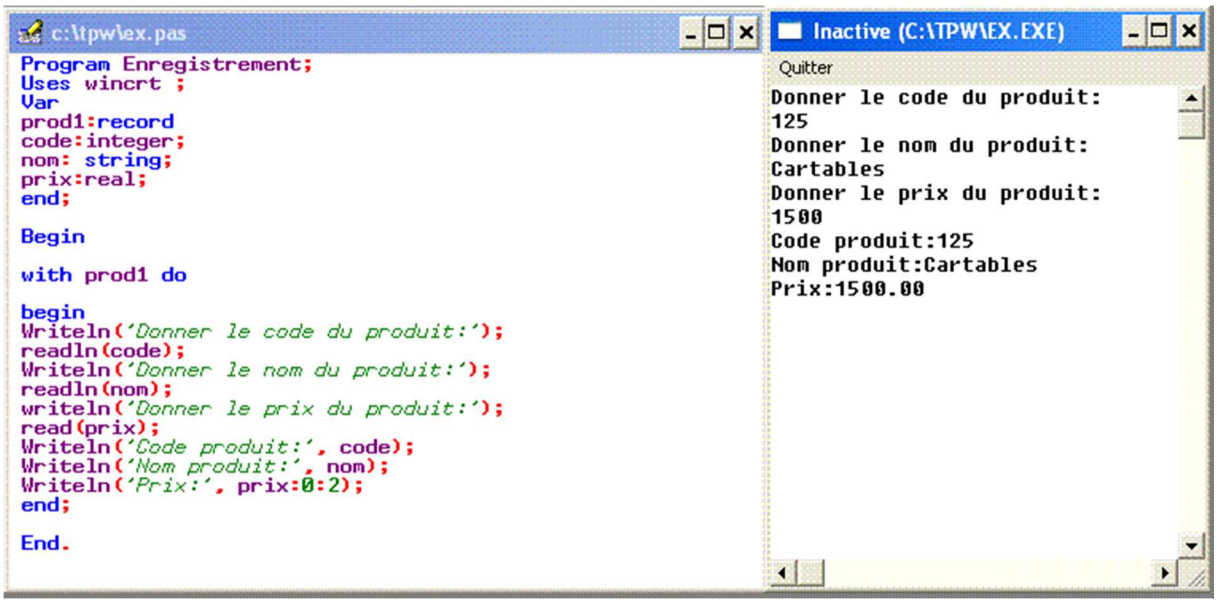
***End ;***

***End ;***

## 10.6. Parcourir les enregistrements à l'aide de la structure *With-do*

La structure ***With-do*** permet de parcourir les champs d'un enregistrement sans avoir besoin de préciser le nom de l'enregistrement pour chaque champ.

Le code ci-dessous reprend l'exercice d'application avec la structure ***With-do***



# 11. TP N°10-Les procédures et les fonctions-

## 11.1. Objectifs

A la fin de cette séance, l'étudiant sera capable de :

- Définir ce que c'est une procédure et une fonction
- Faire la différence entre une procédure et une fonction.
- Déclarer une procédure.
- Appeler une procédure depuis un programme principal.
- Déclarer une fonction.
- Appeler une fonction.

## 11.2. Définition d'une procédure et d'une fonction

Une procédure ou une fonction est un sous-programme écrit avant le corps du programme principal mais qui peut être appelé depuis ce programme principal ou à partir d'une autre procédure et/ou d'une autre fonction.

Les procédures et les fonctions sont écrites lorsqu'un traitement particulier se répète plusieurs fois dans le programme principal. Ainsi, au lieu de réécrire le code du traitement à chaque fois, il suffit de remplacer ce code par le nom de la procédure ou de la fonction. De ce fait, le programme devient plus lisible et facile à corriger.

## 11.3. Différence entre une procédure et une fonction

Une procédure est un sous-programme qui permet de réaliser un ensemble d'instructions mais qui ne renvoie pas de valeurs tant dis que la fonction est utilisée pour calculer et retourner une valeur dont le type est connu au moment de sa déclaration.

## 11.4. Déclarer une procédure

La déclaration se fait toujours avant le début de la section exécutable du programme principal selon la syntaxe suivante :

| | |
|---|---|
| ***Program programme_principal;*** | {Entête du programme principal} |
| ***procedure nom_procedure (nom_variable :type ;…) ;***<br>***begin***<br>***…***<br>***instructions***<br>***…***<br>***end;*** | {Déclaration de la procédure} |
| ***begin***<br>***…***<br>***End.*** | {Corps du programme principal} |

**Exemple**

Le code suivant permet de déclarer une procédure nommée « permuter » qui permet de permuter les valeurs de deux variables x et y de type réel.

***Procedure permuter (var x, y :real) ;***

***Var***

***t : real ;***

***Begin***

***t :=x ;***

***x :=y ;***

***y :=t ;***

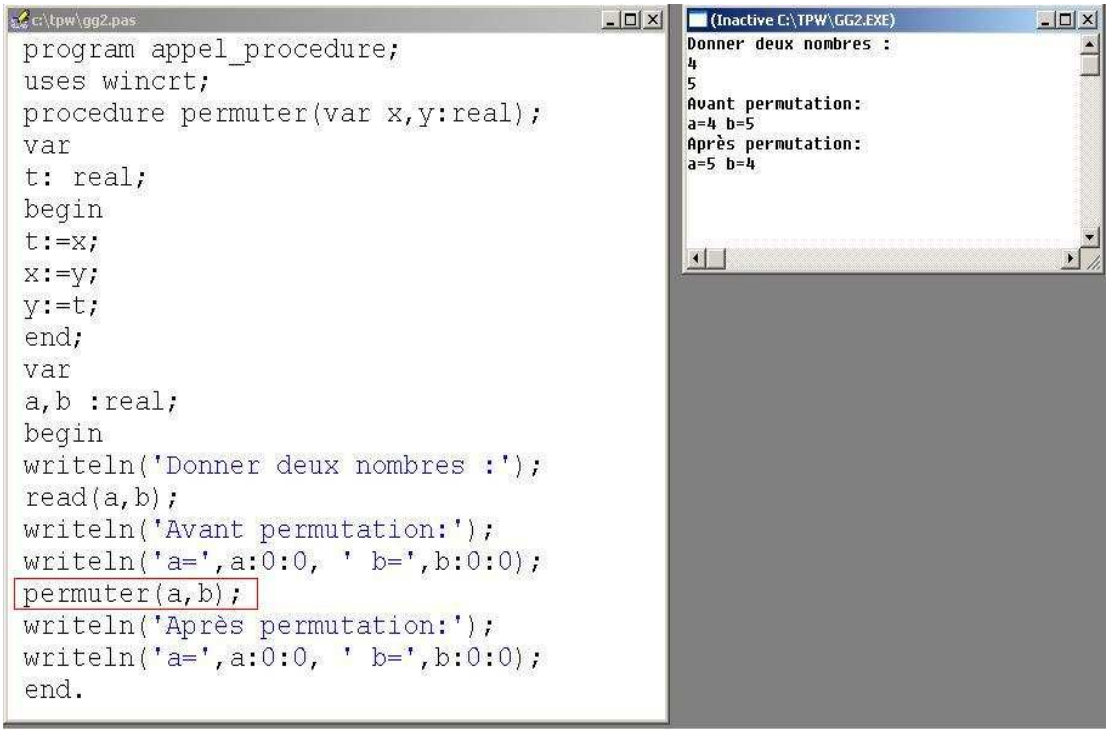
***end ;***

## 11.5. Appeler une procédure

Pour faire appel à une procédure dans un programme principal, il suffit de mettre son nom avec ses paramètres suivi par un point-virgule.

**Exemple**

Dans le programme suivant, nous allons faire appel à la procédure « permuter » déclarée précédemment.



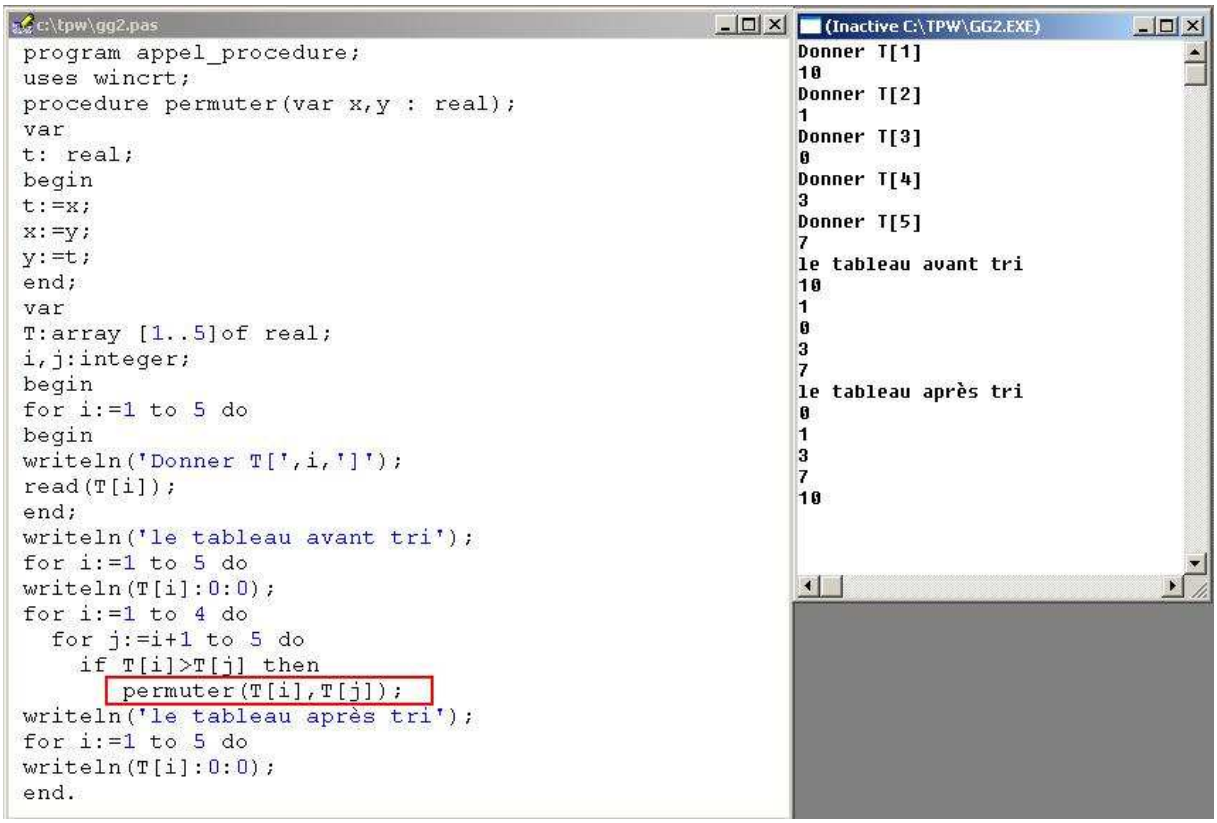

## 11.6. Exercice d'application

Ecrire un programme qui permet de :

1. Remplir un tableau T de 5 éléments de type réel.
2. Trier les éléments du tableau T par ordre croissant en utilisant la procédure « permuter ».

## 11.7. Corrigé de l'exercice d'application

Pour trier les éléments de tableau, nous allons procéder à une série de comparaisons. Chaque élément du tableau est comparé à l'élément qui le suit si ce dernier est plus petit que l'élément sélectionné les valeurs des deux éléments sont permutées.



```
program appel_procedure;
uses wincrt;
procedure permuter(var x,y : real);
var
t: real;
begin
t:=x;
x:=y;
y:=t;
end;
var
T:array [1..5]of real;
i,j:integer;
begin
for i:=1 to 5 do
begin
writeln('Donner T[',i,']');
read(T[i]);
end;
writeln('le tableau avant tri');
for i:=1 to 5 do
writeln(T[i]:0:0);
for i:=1 to 4 do
  for j:=i+1 to 5 do
    if T[i]>T[j] then
      permuter(T[i],T[j]);
writeln('le tableau après tri');
for i:=1 to 5 do
writeln(T[i]:0:0);
end.
```

```
Donner T[1]
10
Donner T[2]
1
Donner T[3]
0
Donner T[4]
3
Donner T[5]
7
le tableau avant tri
10
1
0
3
7
le tableau après tri
0
1
3
7
10
```

## 11.8. Déclarer une fonction

La déclaration se fait toujours avant le début de la section exécutable du programme principal selon la syntaxe suivante :

| | |
|---|---|
| ***Program programme_principal;*** | {Entête du programme principal} |
| ***procedure nom_function (nom_variable :type ;....) :type ;***<br>***begin***<br>***...***<br>***instructions***<br>***...***<br>***end;*** | {Déclaration de la fonction} |
| ***begin***<br>***...***<br>***End.*** | {Corps du programme principal} |

A la différence des procédures, dans la déclaration des fonctions le type du résultat retourné par la fonction doit être précisé.

Exemple

Le code suivant permet de déclarer une fonction « puissance » qui permet de calculer $x^n$.

***Function puissance (x :real ; n :integer) :real ;***

***Var***

***P :real ;***

***i :integer ;***

***Begin***

***P :=1 ;***

***If n>0 then***

***For i :=1 to n do***

***P :=P*x***

***Else***

***For i :=-1 downto n do***

***P :=P/x ;***

***Puissance :=P ;***

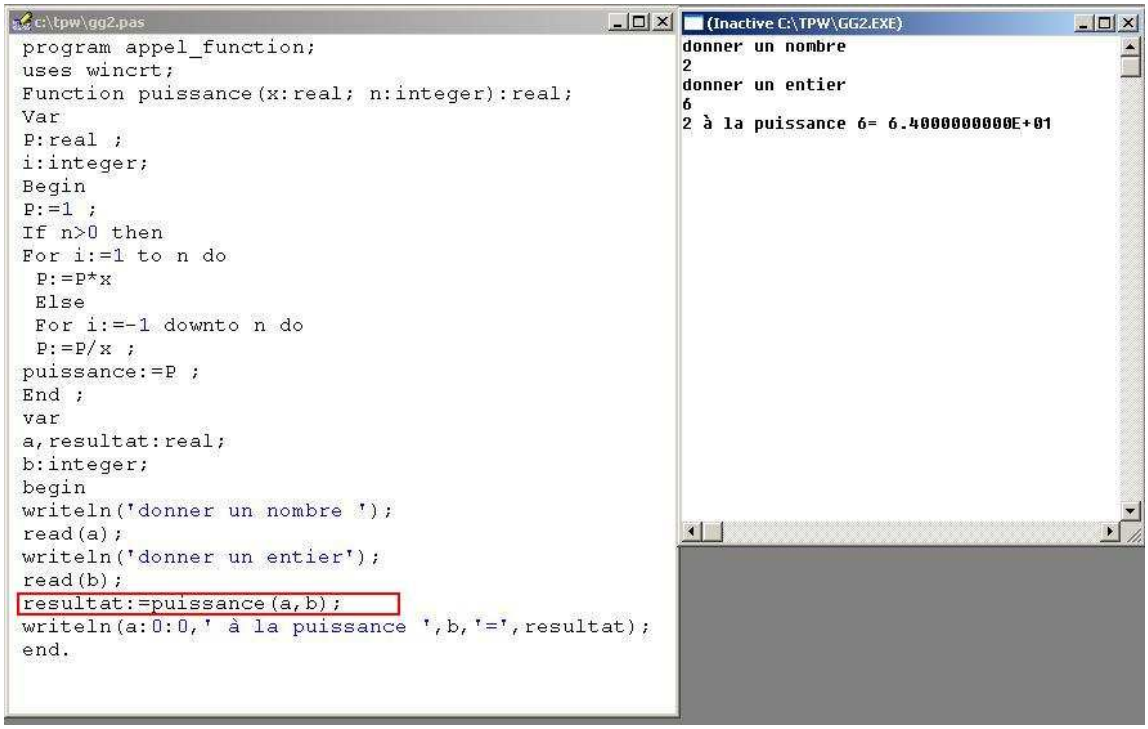
***End ;***

## 11.9. Appeler une fonction

Puisque la fonction retourne toujours une valeur, elle doit être une source d'affectation.

**Exemple**

Dans le programme suivant, nous allons faire appel à la fonction « puissance » déclarée précédemment pour calculer la valeur $a^b$.



## 11.10. Exercice d'application

Ecrire un programme qui utilise la fonction « puissance » pour calculer et afficher la somme suivante : $S=a+a^2+a^3+a^4+....a^m$

## 11.11. Corrigé de l'exercice d'application

c:\tpw\gg2.pas

```
program appel_function;
uses wincrt;
Function puissance(x:real; n:integer):real;
Var
P:real ;
i:integer;
Begin
P:=1 ;
If n>0 then
For i:=1 to n do
 P:=P*x
 Else
 For i:=-1 downto n do
 P:=P/x ;
puissance:=P ;
End ;
var
a,s:real;
m,i:integer;
begin
writeln('donner un nombre ');
read(a);
writeln('donner un entier');
read(m);
s:=0;
for i:=1 to m do
 s:=s+puissance(a,i);
writeln('s=',s);
end.
```

(Inactive C:\TPW\GG2.EXE)

```
donner un nombre
2
donner un entier
4
s= 3.0000000000E+01
```

## 12. Exercices supplémentaires

**Exercice 1**

- Ecrire un programme qui calcule la moyenne de 03 nombres introduits au clavier.

**Exercice 2**

- Ecrire un programme qui calcule le Volume V et la Surface S d'un cylindre de Rayon R et de Hauteur H sachant que :
  Volume d'un cylindre : $V=\Pi.R^2.H$
  Surface d'un cylindre : $S=2.\ \Pi\ .R.H$
  $\Pi$ est une constante = 3.14159

**Exercice 3**

- Ecrire un programme qui calcule et affiche la distance entre 02 points connaissant leurs coordonnées $P_1(x_1,y_1)$, $P_2(x_2,y_2)$.

**Exercice 4**

En utilisant la structure alternative, écrire un programme qui lit deux nombres et affiche le maximum entre eux.
Ecrire le même programme en utilisant la structure conditionnelle**.**

**Exercice 5**

Ecrire un programme qui calcule et affiche l'indice de la masse corporelle (IMC) d'une personne de poids (P) et de taille (T), sachant que : $IMC = \frac{P}{T^2}$. Le programme doit ensuite afficher l'un des messages suivants :

- 'Dénutrition' si IMC<16
- 'Maigreur' si 16≤ IMC <18
- 'Corpulence Normale' si 18≤ IMC ≤25
- 'Surpoids' si 25<IMC<30
- 'Obésité' si IMC ≥ 30

**Exercice 6**

Un grand magasin offre à ces clients la possibilité de bénéficier d'une réduction sur le total des achats effectués. Le taux de réduction est lié au résultat obtenu suite au lancement d'un dé. Il est déterminé comme suit :

- Si le résultat obtenu est 1 alors le taux de réduction=5%
- Si le résultat obtenu est 2 alors le taux de réduction=7%
- Si le résultat obtenu est 3 alors le taux de réduction=8%
- Si le résultat obtenu est 4 alors le taux de réduction=15%
- Si le résultat obtenu est 5 alors le taux de réduction=20%
- Si le résultat obtenu est 6 alors le taux de réduction=50%

En utilisant l'instruction à choix multiple (case), écrire un programme qui lit le montant des achats et le résultat du lancement du dé, calcule et affiche le taux de réduction ainsi que le net à payer.

**Exercice 7**

Ecrire un programme en Pascal qui calcule et affiche la somme suivante :

$$S = -1 + \frac{1}{2} - \frac{1}{3} + \frac{1}{4} - \frac{1}{5} + \cdots + \frac{1}{n}$$

**Exercice 8**

Ecrire un programme en Pascal qui lit une suite de mots se terminant par le mot 'Fin' et calcule et affiche la longueur et le mot le plus long.

**Exercice 9**
Un nombre est parfait s'il est égal à la somme de ses diviseurs stricts (différents de lui-même). Ainsi par exemple, l'entier 6 est parfait car 6 = 1 + 2 + 3. Écrire un programme permettant de déterminer si un entier naturel est un nombre parfait.
Tester le programme avec les nombres : 6, 28, 496, 8128

**Exercice 10**
Ecrire un programme qui permet de :
- remplir deux tableaux d'entiers T1 et T2 de capacité 5.
- calculer et stocker la somme des deux tableaux dans un tableau T3.
- Afficher le contenu du tableau T3.

**Exercice 11**
Ecrire un programme qui permet de :
- Remplir une matrice d'entiers de taille n*m
- Afficher le nombre d'éléments impairs par colonne.

**Exercice 12**
Ecrire un programme qui permet de remplir une matrice M de 3 lignes et 4 colonnes et de lire un nombre x. Le programme doit ensuite calculer et afficher le nombre d'occurrences (répétitions) du nombre x dans la matrice M.

**Exercice 13**
Ecrire un programme qui permet de :

1- Déclarer un type enregistrement nommé « Fournisseur » composé des éléments suivants :
« code » de type entier
« nom » de type chaine de caractère
« adresse » de type chaine de caractère
« CA » de type réel.
2- Déclarer et remplir un tableau d'enregistrements de capacité 4 de type fournisseur.
3- Afficher le nom et l'adresse des fournisseurs ayant un CA supérieur à une valeur donnée.
4- Tester le programme avec les données suivantes :

| | Code | Nom | Adresse | CA |
|---|---|---|---|---|
| 1 | 125 | Mohamed | Oran | 125000,236 |
| 2 | 223 | Rachid | Alger | 235100,453 |
| 3 | 441 | Reda | Mostaganem | 10000,235 |
| 4 | 954 | Kamel | Annaba | 25000,236 |

**Exercice 14**
Ecrire un programme qui utilise une fonction « fact » qui calcule le factoriel d'un nombre entier.
Utiliser la fonction fact pour calculer la somme

$$s = 1 + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \cdots .. + \frac{1}{n!}$$

# 13. Solutions des exercices supplémentaires

**Exercice 1**

```
program  moyenne ;
Uses wincrt;
Var
a, b,c, m : real ;
begin
write  ('donner un nombre') ;
read(a) ;
Write ('donner un 2ème nombre') ;
read(b) ;
Write ('donner un 3ème nombre') ;
read(c) ;
M :=(a+b+c)/3 ;
write ('la moyenne =', m) ;
end.
```

**Exercice 2**

```
Program  cylindre ;
Uses wincrt;
Const
Pi=3.14159 ;
Var
R, H,S , V :real ;
begin
write ('donner le rayon du cylindre') ;
read(R) ;
write ('donner la hauteur du cylindre') ;
read(H) ;
V := Pi*R*R*H ;
S := 2*Pi*R*H ;
write ('la surface=', S);
write ('le volume =', V) ;
end.
```

**Exercice 3**

```
program   distance ;
Uses wincrt;
Var
x1, y1, x2, y2,d : real ;
begin
write('Donner les coordonnées du premier point') ;
read(x1, y1) ;
write('Donner les coordonnées du 2ème points') ;
read(x2, y2) ;
d :=sqrt((x2-x1)* (x2-x1)+(y2-y1)* (y2-y1)) ;
write ('La distance entre les deux points =', d) ;
end.
```

**Exercice 4**

**Avec la structure alternative**

```
program Nombre_Maximum ;
Uses wincrt;
Var
A,B,max : real ;
Begin
write (' Saisir les valeurs de A et B  ') ;
read (A, B)
if (A>=B) then
  max :=A
 else
   max :=B ;
end.
```

**Avec la structure conditionnelle**

```
Program  Nombre_Maximum ;
Uses wincrt;
Var
A,B, max : real ;
begin
write (' Saisir les valeurs de A et B  ') ;
read (A, B) ;
Max :=A ;
if (max< B) then
Max:=B;
write (' le max est : ',max) ;
end.
```

**Exercice 5**

```
Program   ex4 ;
Uses wincrt;
Var
p,t, imc :real ;
Begin
Write('Donner le poids de la personne') ;
Read(p) ;
Write('Donner la taille de la personne') ;
Read(t) ;
imc :=p/(t*t) ;
Write('imc=',imc) ;
If(imc<16) then
  Write('Dénutrition')
   Else
      If(imc<18)  then
         Write('Maigreur')
          Else
              If(imc<=25) then
                Write('Corpluence Normale')
                        Else
                            If(imc<30) then
```

```
                    Write('Surpoids')
                     Else
                       Write('Obésité);
End.
```

**Exercice 6**

```
Program exo6;
Uses wincrt;
Var
mnt,  net :real ;
tred,d:integer;
begin
write('Donner le montant des achats') ;
read(mnt) ;
write('Donner le résultat du lancement du dé') ;
read(d) ;
case d  of
1 : tred :=5 ;
2  : tred :=7 ;
3  : tred :=8 ;
4  : tred :=15 ;
5  : tred :=2 ;
6  : tred :=5 ;
End;
Write('Le taux de réduction=',tred) ;
Net :=mnt*red/100 ;
Write('le net à payer=',net) ;
End.
```

**Exercice 7**

**Avec la boucle for**

```
Program  exo7 ;
Uses wincrt ;
Var
n, i : integer ;
s : real;
Begin
Writeln('donner un nombre entier naturel');
Read(n) ;
s:=0;
For i :=1 to n do
If (i mod 2=0) then
  s:=s+1/i
else
s:=s-1/i;
writeln('La somme=',s);
end .
```

**Avec la boucle while**

```
Program  exo7 ;
Uses wincrt ;
Var
n, i : integer ;
s : real;
Begin
Writeln('donner un nombre entier naturel');
Read(n) ;
s:=0;
i:=1;
while (i<=n) do
Begin
  If (i mod 2=0) then
  s:=s+1/i
  else
  s:=s-1/i;
i:=i+1;
end;
writeln('La somme=',s);
end .
```

**Avec la boucle Repeat**

```
Program  exo7 ;
Uses wincrt ;
Var
n, i : integer ;
s : real;
Begin
Writeln('donner un nombre entier naturel');
Read(n) ;
s:=0;
i:=1;
if (n>0) then
Repeat
  If (i mod 2=0) then
  s:=s+1/i
  else
  s:=s-1/i;
i:=i+1;
until (i>n);
writeln('La somme=',s);
end .
```

**Exercice 8**

**Avec la boucle while**

```
program test;
uses wincrt;
var
long, mot: string;
l:integer;
begin
long:='';
l:=0;
writeln('donner un mot');
readln(mot);
while (mot<>'fin') do
begin
 if(length(mot)>l) then
   begin
   long:=mot;
   l:=length(mot);
   end;
writeln('donner un mot');
readln(mot);
end;
writeln('le mot le plus long =',long);
writeln('sa longueure est=', l);
end.
```

**Avec la boucle Repeat**

```
program test;
uses wincrt;
var
long, mot: string;
l:integer;
begin
long:='';
l:=0;
writeln('donner un mot');
readln(mot);
if (mot<>'fin') then
Repeat
 if(length(mot)>l) then
   begin
   long:=mot;
   l:=length(mot);
   end;
writeln('donner un mot');
readln(mot);
until (mot='fin');
writeln('le mot le plus long =',long);
writeln('sa longueure est=', l);
end.
```

**Exercice 9**

```
Program exo9;
Uses wincrt ;
```

```
Var
s, n, i : integer ;
Begin
Writeln('donner un nombre entier naturel');
Read(n) ;
s:=0;
for i:=1 to n-1 do
 if(n mod i=0) then
   s:=s+i;
if (s=n) then
 writeln(n,' est parfait')
 else
 writeln(n, ' n''est pas parfait');
end .
```

**Exercice 10**

```
program exo1;
uses wincrt;
var
t1,t2,t3: array[1..5] of integer;
i: integer;

begin
for i:=1 to 5 do
begin
writeln('Donner l''élément numéro', i, ' du tableau T1');
read(t1[i]);
end;

for i:=1 to 5 do
begin
writeln('Donner l''élément numéro', i, ' du tableau T2');
read(t2[i]);
end;

for i:=1 to 5 do
begin
t3[i] :=t1[i]+t2[i] ;
writeln('L''élément numéro', i, ' du tableau T3=', T3[i]);
end;

end.
```

**Exercice 11**

```
Program exo6 ;
Uses wincrt ;
Var
Mat:array [1..100,1..100] of integer ;
n,m,i,j:integer;
tab:array[1..100] of integer;
begin
writeln('Donner le nombre de lignes de la matrice');
```

```
read(n) ;
writeln('Donner le nombre de colonnes de la matrice');
read(m) ;
For j:=1 to m do
Begin
  Tab[j]:=0;
  For i:=1 to n do
  begin
     Writeln('Donner mat[',i,'][',j,']');
     Read(mat[i][j]);
     If (mat[i][j] mod 2=0) then
     Tab[j]:=tab[j]+1;
  End;
End;

For j:=1 to m do
  Writeln('Le nombre d''éléménts pairs de la colonne n°', j,'=',tab[j]);
End.
```

**Exercice 12**

```
 Program exo3 ;
 var
 oc, i, j:integer ;
 x: real
 M : array [1..3,1..4] of real ;
 begin
 for i := 1 to 3 do
  for j := 1 to 4 do
 begin
   write('M[', i ,'][' , j , ']=');
   read( M[i,j] ) ;
 end;
 write ('Donnez le nombre x');
 read(x) ;
 oc:=0;
 for i := 1 to 3 do
  for j := 1 to 4 do
   if(M[i,j] =x) then oc:=oc+1;
 write ('le nombre d'occurrences du nombre ',x,'=',oc);

 end.
```

**Exercice 13**

```
Program exo ;
Uses wincrt ;
type fournisseur=record
code : integer;
nom: string;
adresse: string;
ca:real;
end;
```

```
var
t:array[1..4] of fournisseur;
i:integer;
val:real;
begin
for i:=1 to 4 do
begin
writeln('Donner les information du fournisseur n°',i);
write('Code:');
readln(t[i].code);
write('Nom:');
readln(t[i].nom);
write('Adresse:');
readln(t[i].adresse);
write('CA:');
readln(t[i].ca);
end;
Writeln('Donner la valeur du CA');
read(val);
Writeln('Les fournisseurs ayant un CA > à ', val, ' sont:');
for i:=1 to 4 do
if (t[i].ca>val) then
  writeln('Nom: ', t[i].nom, ' Adresse: ', t[i].adresse);
end.
```

**Exercice 14**

```
Program ex14 ;
Uses wincrt ;
function fact (k :integer):integer ;
var
f, i: integer;
begin
f:=1;
for i:=1 to k do
  f:=f*i;
fact:=f;
end;
var
i, n : integer;
s: real;
begin
writeln('Donner un entier naturel');
read (n);
s:=0;
for i:=1 to n do
  s:=s+1/fact(i);
writeln('S=',s);
end.
```

# Bibliographie


- Mohand Cherif BELAID. *Algorithmique & Programmation en PASCAL, Cours, Exercices, Travaux Pratiques Corrigés*. Pages Bleues Internationales, 2008.

- Claude Bauer et Pierre Vincenti, *Le langage Pascal appliqué à l'algorithmique : cours & exercices corrigés.* Ellipses, 2005.

- I.MALTSEV et M.ANDRé CHAUVIN, *Algorithmes et Fonctions Récursives*, office des publications universitaires, 1980.

- Mounira Belmesk, *Algorithmes et structures de données*, Khawarysm, 1991.

- Patrick Cousot, *Algorithmique et programmation en PASCAL : Exercices et corrigés*, Berti, 1993
- Jacques Jorda et Abdelaziz M'zoughi. *Architecture de l'ordinateur : cours + exos corrigés*. Dunod, 2012.