Module: Informatique
Année universitaire : 2016/2017
Enseignant : DJOUABRI .A
Site WEB: djouabri.com
Email: djouabri@djouabri.com

…… Programmation en langage Pascal …01…

1 Premiers programmes

1.1 Le programme bonjour

Un programme est une suite d'instructions, certaines étant des mots clés.
Ce programme affiche la chaine de caractères < Bonjour > à l'écran :

```
PROGRAM bonjour;
BEGIN
    writeln ('Bonjour');
END.
```

Le compilateur est un logiciel qui lit (analyse) un programme et le traduit en code machine, directement exécutable par le processeur de l'ordinateur.

1.2 Commentaires dans un programme

On place un {commentaire} dans un programme au-dessus ou à cote d'une instruction.
Le commentaire n'est pas pris en compte à la compilation. Il sert à rendre le programme plus clair à la lecture, à noter des remarques, etc. :

```
{ Edouard Thiel - 21/01/2003 }
PROGRAM bonjour;
BEGIN
    {Affiche Bonjour à l'écran}
    writeln ('Bonjour');
END.
```

1.3 Utilisation d'une variable entière

Une variable est une zone dans la mémoire vive de l'ordinateur, dotée d'un nom et d'un type. Le nom de la variable permet d'accéder au contenu de la zone mémoire; le type spécifie la nature de ce qui peut être stocké dans la zone mémoire (entier, réel, caractère, etc.).
On a coutume de représenter une variable par une boite ; dessous on met le nom, au-dessus le type, et dans la boite le contenu.
Exemple avec une variable de nom a et de type entier :

```
PROGRAM var_entiere;
VAR
    a : integer; { D_eclaration }
BEGIN
    a := 5; { Affectation }
    writeln ('valeur de a = ', a); {Affichage : a = 5}
END.
```

La structure de ce programme est en 3 parties : le nom du programme, la partie déclarations, et le corps du programme, qui est une suite d'instructions.
La partie déclaration crée les variables (les boites) ; leur contenu est indéterminé (on met un ' ?' dans chaque boite). La taille de la zone mémoire de chaque variable est adaptée au type (par exemple 1 octet pour un caractère, 4 octets pour un entier, etc).

1.4 Trace et tableau de sortie

La trace d'un programme est obtenue en plaçant des **writeln** pour que le programme affiche les valeurs des variables a l'exécution. Cela sert pour mettre au point un programme en TP.

Le tableau de sortie d'un programme est un tableau avec une colonne par variable, ou l'on écrit l'évolution des variables pendant le déroulement du programme. Demandé en TD et examen.

1.5 Lecture au clavier d'une valeur

```
PROGRAM lit_ecrit;
VAR
    a : integer;
BEGIN
    write ('Entrez un entier : '); { pas de retour chariot }
    readln (a); { Lecture }
    writeln ('valeur de a = ', a);
END.
```

2 Identificateur

Sert à donner un nom _a un objet.

Syntaxe

On appelle lettre un caractère de 'a'..'z' ou 'A'..'Z' ou '_'.

On appelle digit un caractère de '0'..'9'.

Un identificateur Pascal est une suite de lettres ou de digit accoles, commençant par une lettre.

Exemples

x, y1, jour, mois, annee, NbCouleurs, longueur_ligne.

Remarques

. Il n'y a pas de différence entre minuscules et majuscules.

. On n'a pas le droit de mettre d'accents, ni de caractères de ponctuation.

. Un identificateur doit être différent des mots clés (begin, write, real, . . .)

On se sert des identificateurs pour : le nom du programme, les noms de variables, les noms de constantes, les noms de types.

3 Types prédéfinis

Un type décrit un ensemble de valeurs et un ensemble d'opérateurs sur ces valeurs.

Entiers

Déclaration :

VAR *variable1*, *variable2*, ..., *variableN :* **INTEGER***;*

Opérations sur entiers :

- + (addition)
- - (soustraction)
- * (multiplication)
- div (division)
- mod (reste de la division)

Elles sont toutes à résultat entier et nécessitent deux arguments entiers.

Les entiers sont compris entre -(MAXINT + 1) et +MAXINT, qui est une constante standard prédéfinie (sa valeur dépend par contre du compilateur, 32767 pour Turbo Pascal).

Réels

Déclaration :

VAR *variable1*, *variable2*, ..., *variableN* : **REAL**;

Opérations :

- + (addition)
- - (soustraction)
- * (multiplication)

- / (division)

Quand une opération comprend un argument réel et un entier, le résultat est réel. / donne toujours un résultat réel, même si les deux arguments sont entiers.
* et / sont de priorité supérieure à + et -, mais entre * et / tout dépend du compilateur (en général de gauche à droite). En cas d'ambiguïté, utilisez des parenthèses (il n'y a aucun inconvénient à mettre plus de parenthèses que nécessaire).

Exemples d'expressions numériques (soit A = 3, B = 4, C = 2) :

- A + B / C = A + (B / C) = 5
- (A + B) / C = 3.5
- A / B * C = (A / B) * C (1.5) dans certains cas, A / (B * C) (0.375) dans d'autres
- A / BC = valeur de A sur valeur de la variable de nom BC et non A sur B * C
- B * A - 5 * C = (B * A) - (5 * C) = 2

Booléens
Utilisé pour les expressions logiques.
Deux valeurs : **false** (faux) et **true** (vrai).
Opérateurs booléens **: not (négation), and (et), or (ou).**
Déclaration :
VAR *variable1*, *variable2*, ..., *variableN* : **BOOLEAN**;
Ces variables peuvent prendre soit la valeur **TRUE** (vrai), soit la valeur **FALSE** (faux).

Opérations booléennes :

- **AND**
- **OR**
- **NOT**
- **XOR (ou exclusif)**

Ces opérations nécessitent des arguments booléens.

Exemple

```
{ Declaration }
        petit, moyen, grand : boolean;
{ Instructions }
        petit := false;
        moyen := true;
        grand := not (petit or moyen);
```

Table de vérité de ces operateurs

x	y	not x	x and y	x or y
true	true	false	true	true
true	false	false	false	true
false	true	true	false	true
false	false	true	false	false

Opérations à valeur booléenne :

- > (supérieur)
- < (inférieur)
- >= (supérieur ou égal)
- <= (inférieur ou égal)
- = (égal)

- <> (différent)

Ces opérations comparent tous éléments de type simple (les 2 arguments doivent être de même type, sauf entiers et réels qui peuvent être comparés entre eux), et renvoient un booléen. Les caractères sont comparés suivant l'ordre du code ASCII.

AND (et), OR (ou), NOT (non), sont de priorité supérieure aux précédents et ne peuvent opérer que sur des booléens : A > B et C doit être écrit : (A > B) and (A > C). Les parenthèses sont obligatoires pour ne pas faire en premier B and A.

Caractères

Déclaration :

VAR *variable1*, *variable2*, ..., *variableN* : **CHAR**;

Ces variables contiennent un caractère. Ceux-ci sont classés suivant un ordre précis: le code ASCII, qui suit l'ordre suivant :

- Les chiffres '0' à '9' par ordre croissant;
- Les majuscules 'A' à 'Z' par ordre alphabétique;
- Les minuscules 'a' à 'z'.

Les opérateurs sur les chars sont :

- **ord**(c) numéro d'ordre dans le codage ; ici < code ascii >.
- **chr**(a) le résultat est le caractère dont le code ascii est a.
- **succ**(c) caractère suivant c dans l'ordre ascii , chr(ord(c)+1)
- **prec**(c) caractère précèdent c dans l'ordre ascii.

Remarque Il y a erreur à l'exécution si le caractère n'existe pas.

Exemple

```
PROGRAM caracteres;
VAR
    c, d : char;
    a : integer;
BEGIN
    c := 'F';
    a := ord(c); { 70 }
    writeln ('Le code ascii de ', c, ' est ', a);
    a := 122;
    c := chr(a); { 'z' }
    writeln ('Le caractere de code ascii ', a, ' est ', c);
    c := 'j';
    d := succ(c); { 'k' }
    writeln ('Le caractere suivant ', c, ' est ', d);

END.
```

Dans le code ASCII, chaque caractère possible a un numéro de code. Par exemple A a pour code 65. En déclarant C comme variable caractère, on peut mettre le signe A dans C par C := 'A' ou C := Chr(65). Dans le premier cas, il faut mettre les cotes pour différencier 'A' de la variable A. Pour mettre une cote dans C, on peut faire C := Chr(39) ou C := '''' : la 1ère cote pour dire qu'il va y avoir un caractère, les 2 suivantes qui symbolisent la cote (car une seule cote voudrait dire fin du caractère), la dernière qui signifie fin du caractère.

Exercice Afficher les caractères de code ascii de 32 à 255 (sur l'écran et sur, les résultats sont parfois différents).

Divers

- On peut remplacer **chr**(32) par #32, mais pas **chr**(i) par #i.
- Le caractère apostrophe se note ''''.

Une suite de caractères telle que 'Il y a' est une chaine de caractères ; il s'agit D'un objet de type string, que l'on verra plus loin.

4. Les fonctions standard

On peut utiliser comme une variable des fonctions (qui peuvent soit être connues par le compilateur, soit définies par vous-même). Une fonction est un "module" ou "routine" qui renvoie une valeur au programme. Par exemple, A := **sqrt**(B * C) met dans A la racine carrée de B fois C. B * C est appelé **argument** de la fonction.

Les principales fonctions standard connues par tous les compilateurs sont :

- **ABS** : renvoie la valeur absolue
- **SQR** : renvoie le carré
- **SQRT** : racine carrée
- **EX** : exponentielle
- **LN** : log népérien
- **SIN** : sinus
- **COS** : cosinus
- **ARCTAN** : arc tangente
- **SUCC** : variable énumérée suivante
- **PRED** : précédent
- **ROUND** : arrondi à l'entier le plus proche
- **TRUNC** : partie entière (permet de mettre un réel dans un entier : **trunc**(4.5) = 4)

Comme toute variable, une fonction possède un type (entier, réel,...) défini, et ne peut donc être utilisée que comme une variable de ce type.

5. Instruction

On appelle **instruction simple** soit :

- Une affectation;
- Un appel à une procédure (une procédure est un ensemble d'instructions regroupées sous un nom, par exemple READLN);
- Une structure de contrôle (voir plus bas).

On appelle **instruction composée** le regroupement de plusieurs instructions sous la forme :

```
BEGIN instruction1; instruction2; ... ; instructionN END
```

On ne met pas de ; après BEGIN ni avant END (puisque le ; sépare deux instructions). Par contre, si l'instruction composée est suivie d'une autre instruction, on mettra un ; après le END.

Remarque : La lisibilité du programme sera meilleure en mettant une instruction par ligne, et en décalant à droite les instructions comprises entre un BEGIN et un END :

```
BEGIN
  instruction1;
  instruction2;

  instructionN
END
```

On appelle **instruction** une instruction soit simple soit composée.

6. Structures de contrôle

Nos connaissances actuelles ne nous permettent pas de faire des programmes utilisant la capacité de l'ordinateur de répéter rapidement et sans erreur beaucoup de calculs. Nous allons donc remédier immédiatement à cela.

Chaque structure de contrôle forme une instruction (qui peut donc être utilisée dans une autre structure de contrôle).

Boucle WHILE - DO (tant que - faire)
Structure :
```
WHILE expression booléenne DO instruction
```

Elle permet de répéter l'instruction tant que l'expression (ou la variable) booléenne est vraie.

```
PROGRAM racine_a_deux_decimales (input, output);
VAR
  nombre, racine : REAL;
BEGIN
  writeln('Entrez un réel entre 0 et 10');
  readln(nombre);
  racine := 0;
  WHILE racine * racine < nombre DO
    racine := racine + 0.01;
  writeln('La racine de ',nombre,' vaut à peu près',racine)
END.
```

Il faut noter que si l'expression booléenne est fausse dès le début, l'instruction n'est jamais exécutée (ici si nombre = 0). Attention, Pascal n'initialise pas automatiquement les variables à 0, c'est-à-dire que sans l'instruction racine := 0, le programme risquerait de donner une réponse fausse (racine valant n'importe quoi, il sera en général très supérieur à la racine cherchée).

On ne peut répéter qu'**une seule** instruction. Mais celle-ci peut être simple (comme dans l'exemple précédent) ou composée (begin - end).

Exercice ex_puiss :
Faire un programme qui affiche les puissances de 2 jusqu'à une valeur maxi donnée par l'utilisateur (par multiplication successive par 2).

Boucle REPEAT - UNTIL (répéter - jusqu'à ce que)
Structure :
```
REPEAT
  instruction1;
  instruction2;
  ...etc...
  instructionN
UNTIL condition
```

Les N instructions sont répétées jusqu'à ce que la condition soit vérifiée. Même si la condition est vraie dès le début, elles sont au moins exécutées une fois.

```
PROGRAM jeu_simpliste (input,output);
VAR
  a : integer;
BEGIN
  writeln('Entrez le nombre 482');
  REPEAT
    readln(a)
  UNTIL a = 482;
  writeln('C''est gentil de m''avoir obéi')
END.
```

Quelle que soit la valeur initiale de A (même 482), la question sera au moins posée une fois (plus si vous désobéissez).

Exercice ex_jeu :
Faire un jeu qui demande de trouver le nombre entre 0 et 10 choisi par l'ordinateur (en comptant les coups).
On utilisera la fonction Random(N) (non standard, disponible en Turbo Pascal) qui renvoie un entier entre 0 et N-1 compris, par l'instruction valeur_choisie := Random(11).

Boucle FOR - DO (pour - faire)
Structure :

```
FOR variable := valeur_début TO valeur_fin DO instruction
```

La variable_énumérée (non réelle) prend la valeur_début, et l'instruction est exécutée. Puis elle est incrémentée (on passe à la suivante, c'est-à-dire que, si elle est entière, on ajoute 1), et ce jusqu'à valeur_fin (comprise).

L'instruction sera donc exécutée (valeur_fin - valeur_début + 1) fois. Si valeur_ fin est inférieure à valeur_début, l'instruction n'est jamais exécutée. Cette forme de boucle est utilisée chaque fois que l'on connaît le nombre de boucles à effectuer.

On peut utiliser un **pas dégressif** en remplaçant TO par DOWNTO :

```
for lettre := 'Z' downto 'A' do
  writeln(lettre)
```

Dans cet exemple, on écrit l'alphabet à l'envers (en déclarant lettre du type CHAR).

La variable peut être utilisée (mais pas modifiée) dans l'instruction (simple ou composée). Elle est souvent appelée **indice** de la boucle. Sa valeur est perdue dès que l'on sort de la boucle.

Exercice ex_moy :
Faire un programme qui calcule la moyenne de N nombres. N doit être demandé par un READLN.
(initialiser une variable à 0, y ajouter progressivement chaque note puis diviser par N).

Instruction IF - THEN - ELSE (si - alors - sinon)

Structure :

```
IF condition THEN instruction1 (* CAS 1 *)
 { ou }
IF condition THEN instruction1 ELSE
                           instruction2 (* CAS 2 *)
```

Si la condition est vraie, alors on exécute l'instruction1 (simple ou composée). Sinon, on passe à la suite (cas 1), ou on exécute l'instruction2 (cas 2).

Remarquez qu'il n'y a pas de ; devant le ELSE.

Exercice ex_jeu_bis :
Modifier le jeu précédent (ex_jeu) en aidant le joueur (en précisant si c'est plus ou c'est moins).
L'instruction2 peut être composée ou entre autres être une instruction IF :

```
IF condition1 THEN
    instruction1
ELSE IF condition2 THEN
    instruction2
ELSE IF condition3 THEN
  instruction3
  ......
ELSE instructionN
```

Un ELSE correspond toujours au dernier IF rencontré (mais dont on n'a pas encore utilisé le ELSE).

```
IF cond1 THEN
   IF cond2 THEN
      inst1 { cond1 et cond2 }
   ELSE inst2 { cond1 et pas cond2 }
ELSE IF cond3 THEN
   inst3 { pas cond1 mais cond3 }
ELSE inst4 { ni cond1 ni cond3 }
```

Si on désire autre chose, utiliser BEGIN et END :

```
IF cond1 THEN
BEGIN
  if cond2 then
     inst1
END { le prochain ELSE se rapporte à COND1 puisque l'instruction (composée) suivant THEN
est terminée }
ELSE inst2
```

La structure CASE - OF (cas - parmi)

Elle évite d'utiliser une trop grande suite de ELSE IF.

Structure :

```
CASE expression OF { regardez bien où j'ai mis les ; }
  liste_de_cas1 : instruction1;
  liste_de_cas2 : instruction2;
  .....
  liste_de_casN : instructionN
END
```

L'instruction i sera exécutée si l'expression appartient à la liste_de_casi. Les autres ne seront pas exécutées (on passe directement au END). L'expression doit être de type scalaire (pas de réels).

En Turbo Pascal, on accepte une liste_de_cas particulière qui est ELSE (et doit être placée en dernier), pour prévoir le cas où expression n'appartient à aucun des cas cités au dessus. En MS-Pascal on utilise de même OTHERWISE.

```
CASE a * b OF { avec a et b déclarés entiers }
  0 : writeln('un des nombres est nul');
  1, 10, 100, 1000, 10000 : writeln('le produit est une puissance de 10'); { 100000 est
impossible en Turbo Pascal car supérieur à MAXINT }
END
```

Attention, certains compilateurs n'acceptent pas de passer sur un CASE avec une valeur prévue dans aucune liste de cas.