Plan du cours

1) Éléments de base
2) Présentation du formalisme algorithmique
3) Éléments de base du langage PASCAL
4) Modularité
5) Structures de données statiques

Introduction Modularité

- Lorsque l'on développe un programme complexe, le nombre d'instruction devient important. Il est nécessaire de l'organiser (**Modularité**) Il suffit de regrouper sous un même nom les instructions agissant dans le même but.

- On distingue :
  - les **Procédures** qui réalisent un traitement (lecture d'un complexe, tri du fichier étudiant)
  - les **Fonctions** qui effectuent un calcul et **retournent un résultat**

Introduction Modularité

- Les fonctions et les procédures peuvent être appelées plusieurs fois à partir du programme principal ou à partir d'autres fonctions en recevant à chaque fois des paramètres ayant des valeurs différentes. Les Fonctions et les Procédures sont donc des moyens de réutilisation de portions de code.
- Les Fonctions et les procédures sont parfois appelées des **sous-programmes**

75

Procédure

Définition

- Une procédure permet de définir un traitement autonome, nomme par un identificateur et appelable par cet identificateur a partir du programme principal.
- Il est en particulier utile de définir une procédure lorsqu'un même traitement doit être **effectué à plusieurs reprises** dans le programme.
- L'utilisation de procédures permet de **structurer** un programme et d'augmenter sa lisibilité.
- En Pascal les déclarations de procédures et de fonctions doivent être placées après les déclarations de variables.

76

Syntaxe d'une fonction/Procédure

PROCEDURE <nom_procedure>**(** <liste des paramètres>)
< déclaration des objets locaux de la procedure>
DEBUT
{corps de la procedure}
FIN

- Exemple :

```
Procedure Echange(var A,B : reel) {A et B variables par référence}
Var Temp: reel {Temp variable locale}
Debut
        Temp ← A
        A ← B
        B ←Temp
        Ecrire('Les valeurs de ',A,' et ',B,'ont été échangés.')
Fin
```

77

Appel d'une procédure

Une fois la procédure déclarée, elle peut être utilisée dans le programme principal par un "**appel**" a cette procédure, a partir du programme principal, ou a partir d'une autre procédure.

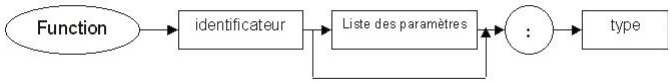
```
Algorithme Classement
Var X, Y : réel

Procedure Echange(var A,B : reel) {A et B variables par référence}
Var Temp: reel     {Temp variable locale}
Debut
         Temp ← A
         A ← B
         B ←Temp
         Ecrire('Les valeurs de ',A,' et ',B,'ont été échangés.')
Fin

Début
         Lire(X,Y)
         Si X>Y alors
                  Echange(X,Y)
         Fin Si
         Ecrire(X, Y)
Fin
```

78

Fonction

Définition

- Une **fonction** est analogue a une procédure mais elle doit de plus retourner une valeur.
- Le type de cette valeur doit être spécifié explicitement.
- Syntaxe : Déclaration
- On déclare les fonctions au même niveau que les procédures (après **const**, **type**, **var**)



79

Syntaxe d'une fonction

FONCTION <nom_fonction> **(** *<liste des paramètres>* **) :** *<type de résultat>*
< déclaration des objets locaux à la fonction>
DEBUT
{ corps de la fonction}
RETOURNER(résultat)
FIN

- Exemple:

```
FONCTION perimetre_rectangle (largeur, longueur :
ENTIER) : ENTIER
   DEBUT
     RETOURNER (2*(largeur+longueur))
   FIN
```

80

Appel de la Fonction

- On appelle une fonction au niveau d'une expression et non d'une instruction comme c'était le cas pour une procédure.
- La fonction est appelée lors de l'évaluation de l'expression et c'est la valeur qu'elle retourne qui est utilisée dans l'évaluation.
- Ainsi, le simple fait d'écrire l'identificateur de fonction avec ses paramètres a pour conséquence d'appeler la fonction, d'effectuer le traitement contenu dans cette fonction, puis de remplacer l'identificateur par la valeur de la fonction dans le calcul.

81

Syntaxe d'une fonction

Exemple : Ecrire un programme qui calcule le cube d'un nombre reel

```
program Puissance3 ;
Var unNombre : real;

function cube (x : real) : real;
begin
       cube:=x*x*x;
       { on affecte la valeur à l'identificateur }
end; { Fin du code de la fonction }

begin { Début du programme principal }
       readln (unNombre);
       writeln ('Le cube de', unNombre,' est : ',
       cube(unNombre) ;
end. { Fin du programme }
```

82

Syntaxe d'une fonction/Procédure

- Exemple: procédure qui fait appel à une fonction

```
FONCTION max3 (x, y, z : ENTIER) : ENTIER
VAR  max :ENTIER

DEBUT
        SI (x < y)
        ALORS_      SI  (z<y )
                    ALORS max ⇦ y
                    SINON  max ⇦z
                    FINSI
        SINON
                    SI (x < z )
                    ALORS max ⇦ z
                    SINON max ⇦ x
                    FINSI
        FINSI
        RETOURNER (max)
FIN
```

83

Syntaxe d'une fonction/Procédure

- Exemple: procédure qui fait appel à une fonction
  - Maintenant, on peut créer la **procédure** qui détermine le maximum et le minimum de trois entiers, en **faisant appel** à la fonction max3 :

```
PROCEDURE  maxETmin(a,b,c: ENTIER)
VAR   min, max : ENTIER
DEBUT
        max ⇦ max3 (a, b, c)
        min ⇦ -max3 (-a, -b, -c)
        ECRIRE ("le minimum de ", a, b, c," est: ", min,"   le maximum est :", max)
FIN
```

84

Différence entre procédure et fonction

- Une procédure peut être considérée comme une instruction composée que l'utilisateur aurait créée lui-même. On peut la considérer comme **un petit programme**.
- Une fonction quant a elle **renvoie toujours une "valeur"**. Elle nécessite donc un type (entier, caractere, booleen, reel, etc...).
  - Remarque
  - Il **est interdit** d'utiliser l'identificateur d'une fonction comme nom de variable en dehors du bloc correspondant a sa déclaration.

85

Variables globales et variables locales

Définition

- Jusqu'a présent, nous avons effectue toutes les déclarations de variables en tête de programme. Or il est également possible de déclarer des variables, **au sein d'un bloc fonction ou procédure.** Dans ce cas, les déclarations se font dans le même ordre : constantes, types, variables (et même procédure(s) et/ou fonction(s)).
- Lorsque la déclaration est effectuée en en-tête du programme, on parle de **variable globale**.
- Dans le cas contraire, c'est-a-dire lorsque la déclaration est faite a l'intérieur même de la procédure ou de la fonction, on parle de **variable locale**.

86

Variables globales et variables locales

- Variable locale
  - la portée est limitée a la procédure ou a la fonction dans laquelle est déclarée et a toutes les fonctions et procédures de niveau inferieur.
- Variable globale
  - active dans le bloc principal du programme, mais également dans toutes les procédures et fonctions du programme.
  - Remarque
    - Les variables déclarées dans un bloc sont actives dans ce bloc et dans tous les blocs de niveau inferieur

87

Variables globales et variables locales

- Porté des variables :

```
procedure NIVEAU-SUP;
var
   X, Y, ...

      procedure NIVEAU-INF;
       var
          X,Z,...
       begin
          X:=...
          Y:=...
          Z:=...
       end;
begin
end.
```

88

Variables globales et variables locales

- La variable X déclarée dans NIVEAU_SUP est **locale** a cette procédure.
- Dans la procédure NIVEAU-INF, cette variable est occultée par l'autre variable X, déclarée encore plus localement.
- La portée de Y est celle de la procédure NIVEAU-SUP, sans restriction (elle est **locale** a NIVEAU-SUP, et donc **globale** dans NIVEAU-INF).
- En revanche, Z a une portée limitée a la procédure NIVEAU-INF (elle est **locale** a NIVEAU-INF, et ne peut être utilisée dans NIVEAU-SUP).

89

Variables globales et variables locales

Remarque

- Dans le cas ou un même nom est utilise pour une variable globale et pour une variable locale, cette dernière a la **priorité** dans la procédure ou dans la fonction ou elle est déclarée (niveau local).
- Dans le cas ou un même nom est utilise pour une variable locale et pour une variable globale, le programme considère ces variables comme **deux variables différentes** (mais ayant le même nom).

90

Variables globales et variables locales

```
program Portee;
Var x : real;

procedure proc1;
Var x : real;
begin
        x:=0;
        writeln (x);
end;

begin { programme principal }
        x:=5;
        proc1;
        writeln(x)
end.
```

A l'exécution, le programme affiche 0, puis 5. En effet, l'appel de proc1 ne modifie pas la variable globale x , puisque x est redéclarée localement a la procédure proc1.

91

Modes de transmission des paramètres

- Il existe deux modes de transmission des paramètres:
- Par **valeur**: le *paramètre transmis n'est jamais affecté* par les modifications dans la procédure ou la fonction (on ne récupère pas les résultats !)
- Par **adresse** (**référence**) : le *paramètre transmis* dans ce cas *peut être modifié* et on récupère les résultats.

92

Passage de paramètre par valeur

Passer a la procédure des valeurs qui seront les données d'entrée et de travail.

Ex.: `procedure ID_PROC (X, Y : real; I : integer; TEST:boolean);`

Points importants

- Les paramètres formels sont des **variables locales** a la procédure, qui reçoivent comme valeurs initiales celles passées lors de l'appel Exemple : `ID_PROC (A, B, 5, true);`
- X a alors pour valeur initiale celle de A, Y celle de B, I a pour valeur initiale 5, et TEST true
- Le traitement effectue dans la procédure, quel qu'il soit, ne pourra modifier la valeur des paramètres effectifs.
- Par exemple : après exécution de ID_PROC, A et B auront **toujours la même valeur qu'auparavant**, même s'il y a eu des changements pour ces variables, dans la procédure. **La transmission est unilatérale**.
- Le paramètre spécifie lors de l'appel peut être une expression. Ainsi,

`ID_PROC (3 / 5, 7 div E, trunc (P), true);` est un appel correct.

93

Passage de paramètre par valeur

Avantages

- les paramètres effectifs peuvent être des expressions.
- les erreurs et les effets de bord sont évites.
- les paramètres sont utilises pour passer des valeurs a la procédure, mais ne sont jamais modifies.

A noter

- Le résultat de l'action accomplie par la procédure n'est pas transmis au programme appelant.
- Si on désire récupérer, le paramètre modifie, il faut alors utiliser un autre procédé, décrit dans la section suivante (passage **par adresse**).

94

Passage de paramètre par valeur

Exemple : Nous cherchons a écrire un programme qui échange les valeurs de deux variables saisies par l'utilisateur.

```
program TEST;
Var A, B : real;
procedure ECHANGE (X, Y : REAL);
Var T : real;
begin
        T := X;
        X := Y;
        Y := T;
        Writeln(X,Y);
end;

begin
        readln (A, B);
        ECHANGE (A, B);
        writeln (A, B);
end.
```

95

Passage de paramètre par valeur

```
program Effet_de_bord;
Var i,j : integer;
function double (i : integer) : integer;
begin
          double := 2 * i;
end;
function plus_un (j : integer) : integer;
Var i: integer;
begin
          i := 10;
          plus_un := j + 1;
end;
function moins_un (j : integer) : integer;
begin
          i := 10;
          moins_un := j - 1;
end;
begin { programme principal }
          i := 0; j := 0; {i=0 ; j=0}
          j := plus_un(i); {i=0 ; j=1}
          j := double(j);{i=0 ; j=2}
          j := moins_un(j);{i=10 ; j=1}
end.
```

96

Passage de paramètre par valeur

- Ce programme a pour seul intérêt d'illustrer le passage de paramètres ainsi que la notion d'effet de bord.
- La variable i a été modifiée dans la procédure, alors que l'on s'attendait a des modifications sur j. On appelle cela un **effet de bord**.
- Un tel phénomène peut être très **gênant** (difficulté a retrouver les erreurs).

97

Passage de paramètre par adresse

- On fournit en paramètre une variable (ou plutôt son adresse) et on travaille directement sur celle-ci, et non sur la valeur contenue dans cette variable.
- Pour réaliser un passage de paramètre par adresse, il faut lors de la déclaration de la procédure (ou de la fonction) ajouter le **mot clé var** devant la déclaration du paramètre concerne. Il est ainsi possible de **récupérer les modifications**
- **effectuées** sur cette variable, a la fin de l'execution de la procédure.
- Exemple : Déclaration

```
procedure ID_PROC (var X, Y : real; Z : integer);
```

98

Passage de paramètre par adresse

- Exemple : `procedure ID_PROC (var X, Y : real; Z : integer);`

Points importants

- Lors de l'appel, des paramètres réels sont substitues aux paramètres formels.
- Tout changement sur le paramètre formel variable change aussi le paramètre effectif spécifie lors de l'appel.
- Seule une variable peut être substituée aux paramètres réels, il est impossible de faire l'appel avec une constante ou une expression évaluable.

- Exemple :
  - `ID_PROC (U, V, 7); { correct }`
  - `ID_PROC (4, A - B, 8); {tout a fait incorrect }`

A noter

- Lorsqu'il y a nécessite de renvoyer une modification au programme appelant, on emploie un passage par adresse.

99

Passage de paramètre par adresse

Exemple

```
program essai;
Var i : integer;
procedure double (x : integer ; var res : integer);
begin
       res := 2 * x;
end;

begin
       i := 1;          { i=1 }
       double (5,i); {i=10}
end;
```

- Dans cet exemple, x est un paramètre transmis par valeur, alors que res est un paramètre passe par adresse.

100

Passage de paramètre par adresse

Exemple : reprenons et corrigeons le programme qui échange les valeurs de deux variables saisies par l'utilisateur.

```
program TEST_BIS;
Var A, B : real;
procedure ECHANGE (var X, Y : real);
Var T : real;
begin
          T := X;
          X := Y;
          Y := T;
          writeln(X,Y);
end;
begin
          readln (A, B);
          ECHANGE (A, B);
          writeln (A, B);
end.
```

Une simulation du déroulement du programme donnera :

- A = 5 B = 7 (saisie)
- X = 7 Y = 5
- A = 7 B = 5 (A et B ont été **modifiés** !)
- Le résultat de l'action de la procédure a été transmis au programme appelant.

101

Passage de paramètre par adresse

```
function LETTRE (c : char) : boolean;
begin
if (c in ['A'..'Z']) or (c in ['a'..'z']) then
          lettre := true
else
          lettre := false;
end;

function MAJ (var c : char) : boolean;
begin
if not LETTRE(c) then
          maj : false
else
          begin
          if c in ['a'..'z'] then
                    begin
                    c := chr (ord(c) - ord('a') + ord('A'));
                    maj := true;
                    end
          else
                    maj := false;
          end;
end;
```

102

Passage de paramètre par adresse

- Ce programme résume ce que nous avons pu voir avec les procédures.
- La première fonction utilise un passage de paramètre par valeur car nous n'avons pas besoin de modifier ce paramètre ; cette fonction est utilisée pour tester si un caractère est une lettre (minuscule ou majuscule).
- La fonction MAJ (qui utilise la fonction précédente) modifie un caractère qui est une lettre minuscule en sa majuscule correspondante.
- Pour les fonctions, on peut agir de même. Le principe est strictement analogue a celui des procédures. On distingue donc, la encore, les passages de paramètres par valeur des passages de paramètres par adresse (ou variables).