Les sous programmes

I. Introduction

L'analyse modulaire a pour objectif de décomposer le problème en sous problèmes (sous programmes, les sous programmes en d'autres sous programmes plus simples et ainsi de suite jusqu'à ce qu'on obtient des sous programmes élémentaires.

On distingue deux types de sous programmes :

Les procédures et les fonctions

II. Les procédures

1. Définition

Une procédure est un sous programme qui permet la résolution d'un problème donné et qui peut avoir plusieurs résultats à partir d'un ou plusieurs paramètres donnés.

2. Syntaxe en algorithme

Algorithme d'une procédure :

0) **Procédure** nom_procédure **(Liste et type des paramètres)**
1) **…**
2) **…**
3) **…**
… …
n) **Fin** nom_procédure

3. Syntaxe en pascal

Procedure nom_procédure **(Liste et type des paramètres) ;**
Type … {Déclaration des types locaux}
Const … {Déclaration des constantes locales}
Var … {Déclaration des variables locales}
Begin
…
…
…
Corps de la procédure
End ;

Remarques :

(1) La définition d'une procédure (ou d'une fonction) se fait toujours dans la partie déclarative du programme principal.

(2) Le programme principal appelle (utilise) les sous programmes définis dans sa partie déclarative : il est dit programme **appelant** et les sous programmes sont dits programmes **appelés**

(3) Les paramètres utilisés au moment de la définition d'un sous programme sont appelés des paramètres formels (fictifs).

(4) Les paramètres utilisés au moment de l'appel d'un sous programme par le programme principal (ou par un autre sous programme) sont appelés des paramètres effectifs (réels).

(5) Les variables locales sont des variables déclarées et utilisée uniquement par le sou programme où elles sont déclarées.

(6) Les variables globales sont des variables déclarées par le programme principal et utilisées par n'importe quel sous programme.

(7) La substitution des paramètres effectifs aux paramètres formels s'appelle **transmission des paramètres**.

(8) L'appel d'une procédure respecte la syntaxe suivante :

Nom_procédure (parmeffect 1, parameffect 2, …, parameffect n)

Avec parameffect 1, …parameffect n sont des paramètres effectifs

(9) Dans l'appel d'un sous programme, il faut respecter le nombre, le type et l'ordre des paramètres effectifs par rapport aux paramètres formels.

(10) En pascal, si les paramètres d'un sous programme sont des types structurés (tableaux), il faut déclarer un nouveau type dans la clause **Type**.

4. Passage des paramètres

Suivant la façon dont on veut qu'il soient passés depuis le corps du programme principal, on peut dans l'entête de la procédure annoncer les paramètres de deux manières :

a) Passage par valeurs :

C'est le processus par défaut :

Lorsqu'un paramètre est passé par valeurs, le compilateur crée une variable locale du même type et lui affecte la valeur du paramètre.

Ainsi une modification dans le corps de la procédure n'affecte que cette variable locale, la modification ne se répercutant pas à l'extérieur de la procédure sur le paramètre passé.

Exemple :

On déclare la procédure :

```
Procedure Inutile (x: real);
        x: = 0;
End;
```

Lorsqu'on appelle la procédure depuis le corps du programme principal sous la forme **Inutile (u) ;** avec u est une variable globale de type réel, tout se passe comme si on déclarait localement : **var x : real ;**

Et si on affectait à x la valeur de u : x :=u ;

La modification de la valeur de x par l'affectation x :=0 ; ne modifie donc pas la valeur de la variable u.

b) Passage par variable (ou par référence)

Dans ce mode de passage, le programme appelé et le programme appelant font un échange de données. En effet, toute modification de la valeur d'un paramètre au sein du programme appelé doit être communiquée au programme appelant. Dans ce mode de passage, on précède le paramètre à transmission par variable par le mot **var**.

Exemple 1 : Initialisation d'un type tableau à 0

```
Type tab=array [1..20] of integer ;

Procedure init_tab (var t:tab);
Var i:integer;
Begin
For i:=1 to 20 do
T[i]:=0;
End;
```

Exemple 2: Procedure sans parameters:

```
Procedure Efface;
Var i:integer;
Begin
For i:=1 to 26 do
Writeln;
End;
```

L'appel de cette procédure se fait par l'instruction simple: **Efface;**

Cette procédure a pour effet de passer 26 fois à la ligne. L'écran étant constitué de 26 lignes, cela revient à l'effacement de l'écran.

Exemple 3 : appeler une procédure depuis une autre

Soit la procédure Ecrit_bonjour déclarée après la procédure efface.

```
Procedure Ecrit_bonjour;
Begin
Efface;
Writeln('Bonjour');
End;
```

III. Les fonctions

1. Définition

Une fonction est un sous programme qui retourne un résultat unique de type simple (entier, réel, booléen, caractère, chaîne de caractères).

2. Syntaxe en algorithme

0) **Fonction** Nom_Fonction (Liste et type des paramètres) : **Type_résultat**
1) …
2) …
… …
n-1) **Nom_Fonction ← Résultat**
n) **Fin** Nom_Fonction

3. Syntaxe en pascal

Function Nom_Fonction **(Liste et type des paramètres) : Type_Résultat ;**
Type … {Déclaration des types locaux}
Const … {Déclaration des constantes locales}
Var … {Déclaration des variables locales}
Begin
…
… } Corps de la fonction
…
Nom_Fonction := Résultat ;
End ;

Remarques :

(1) Les paramètres d'un sous programme ne sont pas des variables, il ne faut donc pas les déclarer.

(2) L'appel d'une fonction peut apparaître partout :

i. Dans une affectation : x :=max (a, b) ;
ii. Dans une comparaison : if pair (a) then …
iii. Dans un affichage : write ('Le maximum est : ', max (a, b)) ;

(3) Il est possible (mais déconseillé) d'utiliser l'identificateur du paramètre d'une fonction pour nommer une variable globale

(4) Il est possible de donner le même identificateur à deux variables, l'une locale et l'autre globale, elles ne seront pas confondues.

(5) Il est possible de déclarer que des variables globales. Mais la déclaration d'une variable globale mobilise (occupe) un espace mémoire pendant toute la durée de l'exécution du programme, tandis que la déclaration d'une variable locale ne mobilise cet espace que le temps de l'exécution du sous programme concerné.

(6) D'une manière générale, la règle à suivre est de **toujours déclarer les variables le plus localement possible.**

Application 1 :

Soit la suite $(U_n)_{n\in N}$ définie par son premier terme $U_0 = \frac{1}{2}$ et pour tout entier n par la relation : $U_{n+1} = 3U_n + 1$

Questions :

1) Écrire un algorithme d'une fonction Eval_U qui permet de retourner le n^{ième} terme de la suite U.
2) Appeler cette fonction dans un programme pascal.

Solution :

1) Algorithme de la fonction Eval_U

Résultat : Calculer le n^{ième} terme de la suite U

Traitement : pour calculer le n^{ième} terme de la suite U, on doit commencer par calculer le second terme, le 3^{ième}, le 4^{ième}, … et le (n-1)^{ième} terme de U.

Donc on peut utiliser une structure itérative complète **Pour…Faire**

U← ½

Pour i de 1 à n Faire
    U ← 3*U + 1
Fin Pour

D'où l'algorithme de la fonction demandée

Paramètre Formel

0) **Fonction** Eval_U (n : entier) : **Réel**
1) U ← ½
2) **Pour** i **de** 1 **à** n **Faire**
   U ← 3*U + 1
   Fin Pour
3) **Eval_U ← U**
4) Fin Eval_U

Sauvegarde du résultat

Tableau de déclaration des objets locaux

Objet	Type/Nature	Rôle
i	Entier	Compteur
U	Réel	Pour stocker le résultat

2) Utilisation de la fonction dans un algorithme principal

0) Début Suite
1) Ecrire ("Entrer le terme à calculer : ") ; Lire (p)
2) Ecrire ('U', p, " = ", Eval_U (p))
3) Fin Suite

Appel de la fonction Eval_U

Tableau de déclaration des objets globaux

Objet	Type/Nature	Rôle
p	Entier	Stocker le terme à calculer sa valeur
Eval_U	Fonction	Pour calculer la valeur du terme p

3) Utilisation de la fonction Eval_U dans un programme pascal

```
program suite;
uses wincrt;
var p:integer;
Function Eval_U(n:integer):real;
var  i:integer; u:real;
begin
u:=1/2;
for i:=1 to n do
u:=3*u+1;
Eval_U:=u;
end;
begin
writeln('entrer le terme à calculer:');
read(p);
writeln('U',p,' = ',Eval_U(p):0:2);
end.
```

Définition de la function

Corps du programme principal

4. Structure générale d'un programme pascal :

```
Program nom_program ;
Uses wincrt ;
Type --- {Déclaration des nouveaux types s'ils existent}
Const --- {Déclaration des constants globales si elles existent}
Var --- {Déclaration des variables globales}
```

Définition d'une fonction

```
Function Nom_Fonction (Listes et types des paramètres) : Type_résultat ;
Type --- {Déclaration des types locaux s'ils existent}
Const --- {Déclaration des constantes locales si elles existent}
Var --- {Déclaration des variables locales s'ils existent}
Begin
---
---                                  } Corps de la fonction
---
Nom_Fonction := Résultat ;
End ;
```

Définition d'une procédure

```
Procedure nom_fonction (Listes et types des paramètres) ;
Type --- {Déclaration des types locaux s'ils existent}
Const --- {Déclaration des constantes locales si elles existent}
Var --- {Déclaration des variables locales s'ils existent}
Begin
---
---          } Corps de la procédure
---
End ;
```

```
Begin
-----
----
----                 } Corps du programme principal
----
----
End ;
```