TD n°3

1.Procédures et fonctions : définition

Quelques procédures et fonctions sont prédéfinies : les procédures « write », « read », les fonctions « cos », « sin », « exp », « ln », « sqrt » (racine), « abs » (valeur absolue), « trunc » (partie entière)... Mais parfois, on a besoin de procédures ou de fonctions qui n'existent pas. Il faut alors les créer.

Mais tout d'abord, quelle est la différence entr'une procédure et une fonction ?

Ce sont toutes deux des sous-programmes. Pour les appeler, on écrit leur nom suivi de leurs paramètres entre parenthèse. Mais une fonction possède un type : elle retourne une valeur de ce type, alors qu'une procédure ne retourne rien et n'est pas typée, mais elle effectue des instructions.

2.Les fonctions

A) Syntaxe

Les fonctions sont insérées après la déclaration des variables et des constantes et avant le begin. Leur syntaxe est très proche de celle d'un programme :

```
Function {nom de la fonction} ( {paramètres de la fonction avec leurs types} ) : {type de la fonction} ;
Var {variables utilisées seulement par cette fonction avec leurs types};
BEGIN
      {instructions}
END;
```

Mais attention :
- Function au lieu de program dans la première ligne,
- Ne pas oublier de typer la fonction (dire si c'est un entier, un réel, ...),
- Les paramètres de la fonction sont séparés par des « ; »,
- Ne pas oublier d'affecter une valeur à la fonction avant de la terminer (voir exemple),
- Après le « END » : un « ; ».

B) Exemple

Le programme suivant affiche les puissances i ièmes d'un réel x, pour i variant de 1 à n. La fonction puissance renvoie $f(x,q)=x^q$, ce qui est un réel.

```
Program serie;
const n=20;
var i:integer;
    somme,x:real;

FUNCTION puissance(x:real;q:integer):real;
var j:integer;
    produit:real;
BEGIN
produit:=1;
for j:=1 to q do
    produit:=produit*x;
puissance:=produit;
END;
```

```
BEGIN
     write('x ? ');
     readln(x);
     for i:=1 to n do
        writeln(puissance(x,i));
     readln;
END.
```

Exercice 1

Ecrire un programme affichant la valeur de $f(x)=\frac{xe^x}{(1+x)^2}$ grâce à une fonction (x donné par l'utilisateur).

Exercice 2

Le but est de calculer $\binom{n}{p}$.

a) Ecrire une fonction retournant n!, puis une autre donnant $\binom{n}{p}$ et utilisant la première.

b) Ecrire une seule fonction qui calcule $\binom{n}{p}$ après simplification de la fraction.

c) Compter le nombre d'opérations * et / de chacune des méthodes. Conclusion ?

C) La fonction random

`x:=random;`
assigne à x un réel aléatoire compris entre 0 et 1, avec une répartition uniforme.

`n:=random(m);`
assigne à n un entier aléatoire compris entre 0 et m-1 avec une répartition uniforme.

Exercice 3

a) Faire un programme retournant le résultat de 10 lancers d'un dé non truqué.
b) Le faire tourner plusieurs fois.
c) Ajouter « randomize; » après le « begin » du programme.
d) Refaire tourner le programme plusieurs fois.

`randomize;`
sert à rendre le tirage réellement aléatoire.

3.Les procédures

A) Syntaxe

Les procédures, comme les fonctions, sont insérées après la déclaration des variables et des constantes et avant le `begin`. L'ordre de déclaration de toutes les fonctions et procédures n'a pas d'importance. Leur syntaxe est à nouveau très proche de celle d'un programme :

`Procedure` *{nom de la fonction}* `(`*{paramètres de la procédure}* `);`
`Var` *{variables utilisées seulement par cette procédure avec leurs types*`};`
`BEGIN`
{instructions}

```
END;
```

Cette fois :

- Procedure au lieu de program,
- Pas de type pour la procédure : elle ne renvoie rien,
- Les paramètres de la procédure sont séparés par des « ; », comme pour une fonction.
- Toujours un « ; » après le « END ».

Une procédure peut utiliser les variables du programme principal non entrées en paramètre, et ayant des noms différents des paramètres. Pour cela, elles doivent être déclarées avant la procédure. Ce sont alors des variables globales (connues du programme et des sous-programmes). La procédure peut même les modifier.

Les variables déclarées à l'intérieur de la procédure sont appelées locales. Elles n'existent pas en dehors de la procédure et sont effacées à la fin de la procédure. Elle doivent porter des noms différents des variables globales (sinon, aucun moyen de les distinguer !).

On peut très bien créer une procédure sans paramètre, n'utilisant que les variables globales et locales. Si l'on donne des paramètres, il y a deux possibilités pour chacun :

- Soit on ne veut pas qu'il soit modifié. Dans ce cas, on le donne comme ceci :

procedure exemple_de_procedure(x:real);

- Soit on veut le modifier lors de la procédure. Alors, on ajoute « var » comme ceci :

procedure exemple_de_procedure(var x:real);

B) Exemple

Le programme suivant permet de demander à l'utilisateur trois entiers entre 3 et 10, grâce à la procédure « saisie » :

```
program exemple_procedure;
var a,b,c:integer;

procedure saisie(var d:integer);
BEGIN
writeln('entrez un entier entre 3 et 10');
readln(d);
while (d<3) or (d>10) do
   begin
      if d<3 then writeln('trop petit')
      else writeln('trop grand');
      writeln('entrez un entier entre 3 et 10');
      readln(d);
   end;
END;

BEGIN
saisie(a);
saisie(b);
saisie(c);
writeln('les trois nombres choisis sont ',a,', ',b,' et ' ,c);
readln;
END.
```

Exercice 4

Faire un programme qui demande deux entiers a et b à l'utilisateur, puis qui utilise une procédure tri(a,b) pour mettre le plus petit des deux dans a et le plus grand dans b. Vérifier que la procédure fonctionne.

Exercice 5

Le but du programme est de calculer la somme partielle $S_n = \sum_{i=1}^{n} \frac{1}{i^2}$.

Faire une procédure incrément(*n*, *S_n*, *u_n)* où *n* est un entier, *S_n* est la somme partielle d'ordre n, et *u_n* est *1/n²*, qui, lorqu'elle est exécutée, transforme *S_n* en *S_n+1*, *u_n* en *u_n+1*.

Utiliser cette procédure pour calculer la somme partielle d'ordre m=20.