TD n°4

1.Array

A) Définition

Pour définir un tableau, on a besoin du type de ses éléments et du rang des indices utilisés pour chacune des dimensions du tableau.

Exemples :

```
VAR tab1:array[0..10]of integer;
    tab2:array[1..3,1..3]of real;
```

Tab1 est un tableau d'entiers à une dimension. Ses éléments sont tab1[0], tab1[1], ... tab1[10].
Tab2 est un tableau de réels à deux dimensions. Ses éléments sont tab2[1,1], tab2[1,2], ... tab2[3,3].

B) Création d'un type

Si l'on utilise un certain type de tableau plusieurs fois dans des fonctions et procédures, on peut créer un « type », qui évite de taper plusieurs fois la définition complète du tableau :

```
TYPE matrice=array[1..3,1..3];
     couple=array[1..2];
VAR M:matrice;
    C:couple;
```

Le squelette d'un programme est maintenant :

```
PROGRAM ...;
CONST ...;
TYPE ...;
VAR ...;
FUNCTION, PROCEDURE ...;
BEGIN
      ...
END.
```

Exercice 1

a) Faire un type pour représenter des nombres complexes.
b) Faire une procédure d'affichage d'un nombre complexe et une pour en entrer un au clavier.
c) Utiliser tout cela dans un programme qui calcule le produit de deux nombres complexes.

Exercice 2

a) Faire une procédure qui permet de remplir un tableau de n fois n (n est une constante) réels.
b) Et une autre qui permet de l'afficher.
c) Tester ces deux procédures.
d) Faire un programme qui ajoute deux matrices et affiche l'opération effectuée.

Exercice 3

On veut calculer le nième terme de la suite $u_n = \sum_{i=1}^{n-1} (n-i)\cos(u_i)$ avec $u_1 = 1$

Faire un programme calculant le nième terme de la suite grâce à un tableau. On prendra un tableau à 100 éléments, et n toujours inférieur ou égal à 100.

Exercice 4 :

On lance un dé traditionnel, à 6 faces, non truqué n fois. On va utiliser un tableau à 6 cases pour stocker le nombre de fois où chaque chiffre a été atteint.

a) Faire une procédure *range(tab)* qui ajoute au tableau *tab* le résultat d'un lancer

b) En faire une autre qui affiche les intitulés des cases du tableau et leur contenu

par exemple : 1 45
2 34...

c) Faire un programme qui effectue 6000 lancers de dé et affiche la quantité de 1,2,... obtenus.

Maintenant, on lance deux dés à la fois, et on utilise un tableau pour stocker la somme du résultat obtenu.

d) Modifier le programme précédent pour afficher le nombre de fois où chaque résultat a été obtenu.

Exercice 5 :

On veut calculer la valeur d'un polynôme $P(x) = \sum_{0}^{n} a_n x^n$ de degré *n* en un point *x* . On représente P par un tableau dont les indices vont de 0 à n, rempli avec les a_i .

Dans notre programme, on a un tableau à 20 cases (de 0 à 19) que l'on remplit au départ de zéros.

Le degré du polynôme est demandé à l'utilisateur (il doit être <20), ainsi que tous les coefficients nécessaires ensuite.

Puis, le polynôme est calculé grâce à une boucle for. Au j ème parcours de la boucle, on ajoute au polynôme le terme de degré j.

a) Faire une procédure qui remplit le tableau de zéros.

b) Puis une qui demande un degré inférieur ou égal à 19, jusqu'à en obtenir un, et un réel x.

c) Enfin, faire le programme qui utilise les deux procédures, remplit le tableau et calcule P(x).

TD n°4 bis

2.Polynômes

On veut calculer la valeur d'un polynôme P, de degré n en un point x :

$$P(x)=\sum_{0}^{n} a_n x^n$$

On représente P par un tableau à une dimension, dont les indices vont de 0 à une constante t, rempli avec les coefficients du polynôme $\{ a_i, i=0..n \}$ pour les n+1 premières cases puis par des zéros. L'idéal serait de prendre t=n, mais il faut donner au programme la dimension du tableau avant de connaître le degré du polynôme... donc, en pratique, on prend un t assez grand, et on n'utilise qu'une petite partie du tableau.

Dans notre programme, on prend un tableau de t+1=20 cases (de 0 à t) que l'on remplit au départ de zéros. Le degré du polynôme est demandé à l'utilisateur (il doit être <20), ainsi que tous les coefficients nécessaires et le réel x pour lequel on veut connaître la valeur du polynôme.

Le programme affiche ensuite P(x).

Exercice 1 : manière naïve

a) Faire une fonction f(x,n) qui calcule x^n .

b) Définir un type « polynome » (tableau de réels à t+1 cases, numérotées de 0 à t).

c) Faire une procédure « vide(P) » qui remplit le tableau P de 0.

d) Faire une procédure « demande_infos(n,P,x) » qui demande à l'utilisateur un degré n, inférieur ou égal à 19, les coefficients correspondants, rangés dans P, et un réel x.

e) Faire une fonction « calc_pol(n,P,x) » qui calcule P(x) grâce à la fonction f.

f) Enfin, taper l'entête et le corps du programme, qui se résume à :

```
begin
    vide(P);
    demande_info(n,P,x);
    writeln('res=',calc_pol(n,P,x));
    readln;
end.
```

g) Enregistrer ! Tester le programme sur des polynômes simples et moins simples.

h) Compter le nombre d'opérations élémentaires (additions, multiplications) effectuées par le programme en fonction du degré du polynôme.

Pour diminuer au maximum le nombre d'opérations élémentaires, il faut simplifier et factoriser au maximum. Par exemple, le calcul suivant : $3x+3y+6x+3z$ est constituée de 7 opérations élémentaires, alors que celui-ci : $3(3x+y+z)$ n'en compte que 4.

Pour les polynômes, la méthode de Hörner permet de réduire ce nombre :

$$P(x)=\sum_{0}^{n} a_n x^n = a_0 + x(a_1 + x(a_2 .....(a_{n-1} + x a_n)).)$$

Description de l'algorithme :

- Si le degré est 0, alors, $P(x)=a_0$.
- Sinon, on donne à une variable réelle K la valeur $a_{n-1}+xa_n$.
- Puis on fait une boucle pour que K devienne $a_{n-2}+xK$, puis $a_{n-3}+xK$, jusqu'à a_0+xK .

Exercice 2 : amélioration

a) Reprendre le programme précédent, ajouter une fonction Hcalc_pol(n,P,x) qui utilise l'algorithme de Hörner.

b) Remplacer calc_pol par Hcalc_pol dans le corps du programme et le tester.

c) Nombre d'opérations élémentaires ? Intérêt de la méthode ?

HORS PROGRAMME DE PREMIERE ANNEE...

3.Les différents tris

On va voir différentes méthodes de tri d'un tableau à n éléments. L'indice des éléments du tableau varient de 0 à n-1.

A) Le tri à bulle

On parcourt le tableau en regardant l'élément i, i variant de 0 à n-2. On compare à chaque fois l'élément i à l'élément i+1. S'ils ne sont pas dans le bon ordre, on les échange. Une fois arrivé au bout du tableau, on est tombé à un moment où l'autre sur le plus grand élément du tableau. On l'a fait remonter par echanges successifs jusqu'à l'emplacement n-1 (sauf s'il s'y trouvait déjà). On n'a donc plus à s'occuper de cet emplacement. On recommence donc, en faisant varier i de 0 à n-3, de façon à faire remonter le deuxième plus grand élément, ainsi de suite, jusqu'à avoir trié complétement le tableau.

Exercice 3

a) Faire une procédure qui permet de remplir un tableau t de n (n est une constante) entiers aléatoires, entre 0 et 100 (0 et 100 inclus).
b) Et une autre qui permet de l'afficher.
c) Faire une précédure tri_bulle(t,n) qui trie ce tableau.
d) Faire un programme qui remplit le tableau, l'affiche, le trie et l'affiche à nouveau.
e) Combien le programme effectue-t'il d'opérations élémentaires (comparaisons, affectations) au maximum, lors du tri ? Au minimum ? Dans quel cas ce maximum est-il atteint ? Et le minimum ?

B) Le tri par insertion

C'est le tri des joueurs de carte. On prend le deuxième élément, et on le met avant ou après le premier selon sa valeur, de façon à avoir les deux premiers éléments du tableau triés. Puis on prend le 3ème, que l'on place par rapport aux deux premiers. Les trois premiers éléments sont alors classés. On continue ainsi jusqu'à ce que tout soit classé.

Exercice 4

a) Reprendre les procédures de remplissage et d'affichage déjà écrites.
c) Faire une précédure tri_insertion(t,n) qui trie ce tableau.
d) Faire un programme qui remplit le tableau, l'affiche, le trie et l'affiche à nouveau.
e) Combien le programme effectue-t'il d'opérations élémentaires (comparaisons, affectations) au maximum, lors du tri ? Dans quel cas ce maximum est-il atteint ?