

CORRIGÉ COLLE D'INFORMATIQUE N°1.

Dans toutes les fonctions que vous avez à écrire, il faut faire apparaître les paramètres d'entrée ainsi que le type de ces paramètres d'entrée. Par exemple, pour la fonction **genere** suivante, **proc(n :: integer)** nous dit que la fonction **genere** a un paramètre d'entrée qui est du type entier. On exclura, sauf mention contraire, l'usage des variables globales.

1. Il faut faire attention au fait que **rand(1..400)** est une fonction et que pour avoir la valeur de cette fonction il faut ajouter «()».

```
> genere := proc(n :: integer)
> local t, i;
> t := array(0..n-1);
> for i from 0 to n-1 do
>   t[i] := rand(0..400)();
> od;
> return(t);
> end;
```

2. On mémorise le plus grand élément dans la variable *m*, on parcourt tout le tableau et l'on compare le *i*ème élément avec *m*. Si le *i*ème élément est plus grand que *m* on stocke sa valeur dans *m* sinon on ne fait rien. En fin de boucle, *m* contient le plus grand élément.

```
> plusgrand := proc(a :: array)
> local n, m, i;
> n := longueur(a);
> m := a[0];
> for i from 1 to n-1 do
>   if a[i] > m then m := a[i] fi
> od;
> return(m);
> end;
```

3. On reprend le programme précédent en ajoutant une variable *k* dans laquelle on stocke l'indice du plus grand élément. Lors du parcours du tableau, si le *i*ème élément est plus grand que *m* on stocke sa valeur dans *m* et *i* dans *k* sinon on ne fait rien. En fin de boucle, *k* contient l'indice du plus grand élément.

```
> indiceplusgrand := proc(a :: array)
> local n, i, m, k;
> n := longueur(a);
> m := a[0]; k := 0;
> for i from 1 to n-1 do
>   if a[i] > m then m := a[i]; k := i fi
> od;
> return(k);
> end;
```

4. Dans le programme **moyenne**, la variable *s* permet de stocker les sommes partielles. À chaque passage dans la boucle, on ajoute le *i*ème élément à *s*. En fin de boucle *s* contient la somme des éléments du tableau *a*. Il ne reste plus qu'à diviser *s* par la longueur du tableau pour avoir la valeur moyenne du tableau.

```
> moyenne := proc(a :: array)
> local s, i, n;
> n := longueur(a);
> s := 0; # initialisation de s
> for i from 0 to n-1 do
>   s := m + a[i]
> od;
> return(evalf(m/n)); # on retourne la valeur moyenne du tableau
> end;
```

5. Pas de difficulté supplémentaire pour cette fonction. Il faut seulement bien faire attention à s'arrêter au bon moment dans la boucle pour ne pas avoir de débordement de tableau.

```
> maxsucc := proc(a :: array)
> local n, i, m;
> n := longueur(a);
> m := abs(a[1]-a[0]);
> for i from 1 to n-2 do
>   if abs(a[i+1]-a[i]) > m then m := abs(a[i+1]-a[i]) fi;
> od;
> return(m)
> end;
```

```

6. > ecrete := proc(a::array, seuil :: integer)
> local n, b, i;
> n := longueur(a);
> b := array(0..n-1);
> for i from 0 to n-1 do
>   if a[i] > seuil then b[i] := seuil else b[i] := a[i] fi;
> od;
> return(b);
> end;

```

7. On remarque tout d'abord que l'amplitude maximale est donnée par la différence entre le plus grand et le plus petit des éléments du tableau. On reprend le principe de la fonction `plusgrand`, en utilisant deux variables, M pour stocker le plus grand élément du tableau et m pour stocker le plus petit.

```

> amplitude := proc(a :: array)
> local n, i, m, M;
> n := longueur(a);
> m := a[0]; M := a[0];
> for i from 1 to n-1 do
>   if a[i] > M then M := a[i]
>   elif a[i] < m then m := a[i] fi;
> od;
> return(M-m);
> end;

```

8. La fonction `croissance_max` demande un peu plus de réflexion et de subtilités. Pour passer de i à $i+1$, on regarde si le taux de croissance entre l'élément courant et le plus petit élément entre les indices 0 et i est plus grand que la meilleure croissance trouvée jusqu'ici et sinon on regarde si le i ème élément ne serait pas plus petit que celui en position `imin`.

```

> croissance_max := proc(a :: array)
> local n, i, croiss, imin;
> croiss := 0; # Pour stocker la meilleure croissance
> imin := 0; # pour stocker le plus petit élément du tableau
> n := longueur(a);
> for i from 1 to n-1 do
>   if a[i] - a[imin] > croiss then croiss := a[i] - a[imin];

```

```

>   # On repère l'indice du plus petit élément
>   elif a[i] < a[imin] then imin := i
>   fi;
> od;
> return(croiss)
> end;

```

9. Programmation de l'algorithme d'Euclide pour trouver les nombres premiers compris entre 2 et n .

```

> premier := proc(n :: integer)
> local i, j, a, b, k, compteur;
> a := array(2..n);
> # on initialise le tableau
> for i from 2 to n do a[i] := true od;
> compteur := 0;
> for i from 2 to n do
>   # si i est un nombre premier,
>   # on met à "faux" les multiples de i
>   if a[i]
>     then
>       # pour compter le nombre de nombres premiers
>       compteur := compteur+1;
>       for j from i to n/i do a[i*j] := false od;
>     fi;
>   od;
> # création du tableau de sortie des résultats
> b := array(1..compteur);
> k := 1;
> # on remplit le tableau des résultats
> for i from 2 to n do
>   if a[i] then b[k] := i; k := k+1 fi;
> od;
> return(b) # on sort le résultat.
> end;
>

```

10. Exponentiation rapide dans sa version récursive :

```
> expor := proc(x, n::integer)
> if n=1 then x
> elif n mod 2 = 0
>   then expor(x,n/2)^2
>   else x*expor(x,(n-1)/2)^2
> fi;
> end;
```

Exponentiation rapide dans sa version itérative :

```
> expoi := proc(x,n::integer)
> local p, m, res;
> m:= n;
> p := x; res := 1;
> while m <> 0 do
>   if m mod 2 = 0
>     then p := p*p; m := m/2;
>     else res := p*res; p:=p*p; m := (m-1)/2
>   fi;
> od;
> return(res);
> end;
```

11. Multiplication du paysan russe : Le principe est le même que celui de l'exponentiation rapide.

Version récursive :

```
> russer := proc(p :: integer, q :: integer)
>   if q = 0 then return(0)
> elif q mod 2 = 0 then return(russer(p+p,q/2))
>   else return(p+russer(p+p,(q-1)/2))
> fi;
> end;
```

Version itérative :

```
> russei := proc(a::integer,b::integer)
> local p, q, res;
> p := a;
> q := b; res := 0;
> while q <> 0 do
>   if q mod 2 = 0
>     then p := p+p; q := q/2;
>     else res := p+res; p:=p+p; q := (q-1)/2
>   fi;
> od;
> return(res);
> end;
```

12. fonction «retourne», les indices du tableaux sont supposés allés de 0 à $n-1$ (où n et la longueur du tableau)

```
>longueur := proc(a :: array)
>   return(nops(convert(a,list)));
> end;

> retourne := proc(a :: array)
> local i, n ,b;
> n := longueur(a);
> b := array(0..n-1);
> for i from 0 to n-1 do
>   b[i] := a[n-1-i];
> od;
> return(b)
> end;
```

Si on veut «retourner sur place» le tableau a , c'est à dire, ne pas créer un nouveau tableau, la fonction s'écrit :

```
# la fonction <<echange>> permute 2 éléments dans un tableau.
> echange := proc(a::array,i::integer, j :: integer)
> local c;
>   c:= a[i]; a[i] := a[j]; a[j] := c;
> end;

> retourne := proc(a :: array)
> local i, n;
> n := longueur(a);
> for i from 0 to (n-1)/2 do
>   echange(a,i,n-1-i);
> od;
> return(a)
> end;
```

13. fonction «EcartType» sur un tableau a indexé de 0 à $n-1$ (où n et la longueur du tableau)

```
>moyenne := proc(a::array)
> local n, s, i;
> n := longueur(a); s := 0;
> for i from 0 to n-1 do s := s + a[i] od;
> return(evalf(s/n));
> end;

> EcartType := proc(a::array)
> local n, m, i, s;
> m := moyenne(a);
> n := longueur(a);
> s := 0;
> for i from 0 to n-1 do s := s + (a[i]-m)^2 od;
> return(sqrt(s/n));
> end;
```