Correction série les boucles

Exercice #1: who

On cherche un nombre $n \in [1000 \text{ à } 9999]$, tel que $n \mid n \text{ à } l'envers$.

Par exemple:

$n = 1089$

$n \text{ à } l'envers = 9801$ } $1089 \mid 9801$

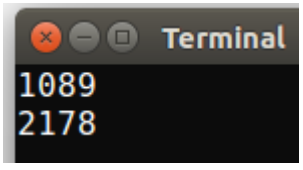
Il existe un autre nombre n de 4 chiffres qui vérifie la même propriété. Si vous essayer avec un papier et un stylo alors bon courage :)

Nous allons coder un programme qui va trouver l'autre nombre en un clin d'oeil ;)

```
(*
* Lang: Free Pascal
* Time complexity: O(9998-1000+1 )
* space complexity: O(1)
*)
var
    n, u, d, c, m: integer;
begin
    for n := 1000 to 9999 do begin
        u := n mod 10;
        d := n div 10 mod 10;
        c := n mod 1000 div 100;
        m := n div 1000;

        if (u <> m) and (d <> c) then
            if (u * 1000 + d * 100 + c * 10 + m) mod n = 0 then
                writeln(n);
    end;
end.
```

Exécution



Le nombre qu'on cherche est: 2178.

Exercice #2: premier

pour chaque $x \in [2, n]$, on cherche s'il y a diviseur d de x, autre que 1 et x. Si on ne trouve pas un diviseur d, alors x est premier.

```
(*
* Lang: Free Pascal
* Time complexity: O(n * (x / 2) )
* space complexity: O(1)
*)
var
	n, x, d: integer;
begin
	repeat
		write('n = ');
		read(n);
	until n > 2;

	for x := 2 to n do begin

		d := 2;
		while (d <= x div 2) and (x mod d <> 0) do
			d := d + 1;

		if d > x div 2 then
			write(x,' ');
	end;
end.
```

Solution proposée par l'élève Akram BOUKHRISS (4e science #1)

```
(*
* Lang: Free Pascal
* Time complexity: O(n * (x / 2) )
* space complexity: O(1)
*)
var
	n, x, d: longint;
	trouverDiviseur: boolean;
begin
	repeat
		write('n = ');
		read(n);
	until n > 2;

	for x := 2 to n do begin
		trouverDiviseur := false;
		for d := 2 to x div 2 do
			if x mod d = 0 then begin
				trouverDiviseur := true;
				break; //quitter la boucle for
			end;
		if not trouverDiviseur then write(x, ' ');
	end;
end.
```

Solution proposée par l'élève Wajdi BEN OUN (4e science #1)

```
(*
* Lang: Free Pascal
* Time complexity: O(n * (x / 2) )
* space complexity: O(1)
*)
var
	n, x, d: integer;
begin
	repeat
		write('n = ');
		read(n);
	until n > 2;

	for x := 2 to n do begin

		d := 1;
		repeat
			d := d + 1;
		until (d > x div 2) or (x mod d = 0);

		if d > x div 2 then
			write(x,' ');
	end;
end.
```

Exercice #3: pronique

pour chaque $n \in [1, 10^9]$, on cherche tout les n qui sont pronique. Pour chaque n, il suffit de trouver x, telque:

$$n = x(x+1) => x^2 + x - n = 0$$

$$x = \frac{-1 + \sqrt{(1+4n)}}{2a}$$

```
(*
* Lang: Free Pascal
* Time complexity: O(109)
* space complexity: O(1)
*)
var
    n, x, cnt: longint;
    rep: char;
begin
    cnt := 0;
    for n := 1 to 1000000000 do begin
        x := trunc((-1 + sqrt(1+4*n)) / 2);
        if n = x * (x+1) then begin
            cnt := cnt + 1;
            writeln(n);
            if cnt mod 10 = 0 then begin
                repeat
                    write('Continuer (O/N): ');
                    readln(rep);
                until (upcase(rep) = 'O') or (upcase(rep) = 'N');
                if upcase(rep) = 'N' then break;
            end;
        end;
    end;
end.
```

Exercice #4: anagramme palindrome.

Une **anagramme** est une construction qui inverse ou permute les lettres d'un mot ou d'un groupe de mots pour en extraire un sens ou un mot nouveau.
Exemple:
crane - écran - nacre - carne - rance - ancre - caner - encra - cerna - caren - créna (du verbe créner) - nerac - narce - nacer - arcen - ceran – renac

Le **palindrome** désigne un texte ou un mot dont l'ordre des lettres reste le même qu'on le lise de gauche à droite ou de droite à gauche, comme dans la phrase « Ésope reste ici et se repose » ou encore « La mariée ira mal » à un accent près.

wikipédia

On vous donne un mot m (contenant que des lettres minuscules non accentuées), vous allez chercher s'il existe une anagramme ce ce mot qui est palindrome.

Méthodhe naive:
*Time complexity: O(n! * longeur(angramme))*
space complexity: O(1)
Pour chaque angramme a de m, vérifier si a est palindrome ou pas.
m = "ananas"
nombre des angrammes = $\frac{6!}{3!2!} = 60$ *angrammes possible*.

nsaaan → non palindrome.
snanaa →non palindrome.
….
dans les 60 angrammes possibles, il a y aucun palidrome.

Imaginez que le nombre des lettres de m est 20, par exemple, sans répititions nous avons $20! = 2.432902008 \times 10^{18}$ lettres à vérifier.

Même l'ordinateur va rester longtemps pour dénombrer tout les angrammes et vérifier pour chacun s'il est palindrome ou pas.

Méthode éfficace:
Puisque notre but est de savoir s'il y a une angramme palindrome et non d'afficher le palindrome lui même.

Si une angramme a est palindrome, nous avons deux cas:

- Le nombre d'occurence de chaque lettre de a est pair.

$$(\sum_{l='a'}^{'z'} \#\mathrm{l} \bmod 2)=0$$

$\#\mathrm{l}: nombre\, d'occurence\, de\, l\, dans\, m.$

- Le nombre d'occurence de chaque lettre de a est pair sauf une lettre où son nombre d'occurence est impair. $(\sum_{l='a'}^{'z'} \#\mathrm{l} \bmod 2)=1$

Donc, si $(\sum_{l='a'}^{'z'} \#\mathrm{l} \bmod 2)<2$ alors il existe une anagramme palindrome dans m.

```
(*
* Lang: Free Pascal
* Time complexity: O(26 * longeur(anagramme) )
* space complexity: O(26 *  sizeof(char))
*)
type
    tab = array['a'..'z'] of integer;
var
    m: string;
    i, somme: integer;
    c: char;
    occ: tab; // Il n'est pas obligatoire d'initaliser le tableau occ à 0
              // Parce qye Free Pascal le fait par défaut.

begin
    //Le mot m doit contenir que des lettres minuscules.
    repeat
        write('donner un mot: ');
        readln(m);

        i := 1;
        while (i <= length(m)) and (m[i] in ['a'..'z']) do
            i := i + 1;
    until i > length(m);

    //Calcul du nombre d'occurence de chaque lettre de m.
    for i := 1 to length(m) do
        occ[m[i]] := occ[m[i]] + 1;

    //Calcul de la somme.
    somme := 0;
    for c := 'a' to 'z' do
        somme += occ[c] mod 2;

    //Vérifier s'il existe une anagramme palindrome.
    if somme < 2 then writeln('OUI')
    else writeln('NO');
end.
```

Exercice #5: display

Il suffit de trouver la formule adéquate.

```pascal
(*
* Lang: Free Pascal
* Time complexity: O(9 )
* space complexity: O(1)
*)
var
    i, n: longint;
begin
    n := 1;
    for i := 1 to 9 do begin
        writeln(n,' * 8 + ', i, ' = ', n * 8 + i);
        n := n * 10 + (i+1);
    end;
end.
```

Exercice #7:

<u>Méthode avec un second tableau</u>

parcourir le premier tableau, si on trouve un nombre pair on le met au dédut du premier tableau, si on trouve un nombre inpair, on le met à la fin du second tabelau.

i= 1

t1:	4	3	13	9	2	1
i:	*1*	*2*	*3*	*4*	*5*	*6*

t2:	4					
	1	*2*	*3*	*4*	*5*	*6*

i= 2

t1:	4	3	13	9	2	1
i:	*1*	*2*	*3*	*4*	*5*	*6*

t2:	4					3
	1	*2*	*3*	*4*	*5*	*6*

i= 3

t1:	4	3	13	9	2	1
i:	*1*	*2*	*3*	*4*	*5*	*6*

t2:	4				13	3
	1	*2*	*3*	*4*	*5*	*6*

i= 4

t1:	4	3	13	9	2	1
i:	*1*	*2*	*3*	*4*	*5*	*6*

t2:	4			9	13	3
	1	*2*	*3*	*4*	*5*	*6*

i= 5

t1:	4	3	13	9	2	1
i:	*1*	*2*	*3*	*4*	*5*	*6*

t2:	4	2		9	13	3
	1	*2*	*3*	*4*	*5*	*6*

i= 6

t1:	4	3	13	9	2	1
i:	*1*	*2*	*3*	*4*	*5*	*6*

t2:	4	2	1	9	13	3
	1	*2*	*3*	*4*	*5*	*6*

l'idée est de maintenir deux indices ip et ii.
L'indice ip pour insérer les nombres pair dans $t2$.
L'indice ii pour insérer les nombres impair dans $t2$.

```pascal
(*
* Lang: Free Pascal
* Time complexity: O(N)
* space compexity: O(2N)
*)

type
    tab = array[1..20] of integer;
var
    n, i, ip, ii: integer;
    t1, t2: tab;
begin
    repeat
        write('n = ');
        read(n);
    until (n >= 2) and (n <= 20);

    randomize;
    for i := 1 to n do
        t1[i] := random(550-55+1)+55;

    for i := 1 to n do
        write(t1[i], ' ');
    writeln;

    ip := 1;
    ii := n;

    for i := 1 to n do
        if t1[i] mod 2 = 0 then begin
            t2[ip] := t1[i];
            ip := ip + 1;
        end
        else begin
            t2[ii] := t1[i];
            ii := ii - 1;
        end;

        t1 := t2; //Free Pascal
        for i := 1 to n do
            write(t1[i], ' ');
    writeln;

end.
```

Méthode in place

on arrange les nombres sur place.

Parcourir le tableau *t*, si on rencontre un nombre pair, on le permute avec le nombre qui se trouve à la position d'insertion

exemple:

Le nombre 18 à la postion 5 est pair.

t:	4	3	13	9	18	1
i:	*1*	*2*	*3*	*4*	*5*	*6*

t:	4	18	13	9	3	1
i:	*1*	*2*	*3*	*4*	*5*	*6*

```pascal
(*
* Lang: Free Pascal
* Time complexity: O(N)
* space compexity: O(N)
*)
type
    tab = array[1..20] of integer;
var
    n: integer;
    t: tab;
    i, j, tmp: integer;
begin
    repeat
        write('n = ');
        read(n);
    until (n >= 2) and (n <= 20);

    randomize;
    for i := 1 to n do
        t[i] := random(550-55+1)+55;

    for i := 1 to n do
        write(t[i], ' ');
    writeln;

    j := 0;
    for i := 1 to n do
        if t[i] mod 2 = 0 then begin
            j := j + 1;
            tmp := t[i];
            t[i] := t[j];
            t[j] := tmp;
        end;

        for i := 1 to n do
            write(t[i], ' ');
    writeln;

end.
```

Exercice #8 suite

```
(*
* Lang.: Free Pascal
* Time complexity: O(n)
*)

var
    n, u0, un, i: longint;
begin
    repeat
        write('N = ');
        read(n);
    until n >= 0;

    u0 := 1;
    for i := 1 to n do begin
        un := 2 * u0 + 1;
        u0 := un;
    end;

    writeln('U(', n, ') = ', u0);
end.
```

(il y a une solution O(1) sans l'utilisation d'une boucle pour cette suite, mais elle ne sera pas abordée parce que les opérateurs utilisés ne sont pas programmés dans votre programme officiel)
Pour les curieux :
https://www.hackerrank.com/challenges/utopian-tree/forum

Exercice #9: Nombre des carrés

```pascal
(*
* Lang.: Free Pascal
* Time complexity: O(sqrt(b))
*)

var
    i, a, b, ans: longint;
begin
    repeat
        write('(a, b) = ');
        read(a, b);
    until (a >= 1) and (b <= 1000000000) and ( a <= b);

    i := 1;
    ans := 0;

    while sqr(i) <= b do begin
        if sqr(i) >= a then
            ans := ans + 1;
        i := i + 1;
    end;

    writeln(ans);
end.
```

(il y a une solution O(1) sans l'utilisation d'une boucle pour cette exercice, mais elle ne sera pas abordée parce que les fonctions utilisées ne sont pas programmées dans votre programme officiel)
Pour les curieux :
https://www.hackerrank.com/challenges/sherlock-and-squares/editorial

Exercice #10: devinette

```
(*
* Programme interactif.
* Lang: Free Pascal.
*)
var
    n, x, nbTent: byte;
begin
    randomize;
    n := random(11)+50;

    writeln('----------------------------');
    writeln('--------Devinette----------');
    writeln('----------------------------');
    writeln('J''ai choisi un nombre entre 50 et 60.');
    writeln('Essayez de le deviner');
    writeln('Vous avez trois tentatives.');
    writeln;

    nbTent := 0;
    repeat
        nbTent := nbTent + 1;
        write(chr(9)+'Tentative #', nbTent,': Taper un nombre: ');
        read(x);
    until (n = x) or (nbTent >= 3);

    writeln;

    if n = x then
        writeln('Bravo :)');
    if (n <> x) then
        writeln('Désolé :/, le numéro à deviner était: ', n,'.');
end.
```

```
----------------------------
--------Devinette----------
----------------------------
J'ai choisi un nombre entre 50 et 60.
Essayez de le deviner
Vous avez trois tentatives.

        Tentative #1: Taper un nombre: 55
        Tentative #2: Taper un nombre: 56
        Tentative #3: Taper un nombre: 58

Désolé :/, le numéro à deviner était: 59.
```

```
----------------------------
--------Devinette----------
----------------------------
J'ai choisi un nombre entre 50 et 60.
Essayez de le deviner
Vous avez trois tentatives.

        Tentative #1: Taper un nombre: 56
        Tentative #2: Taper un nombre: 57
        Tentative #3: Taper un nombre: 51

Bravo :)
```

Exercice #11: Euler

Il s'agit de trouver une valeur approchée de π en utilisant la formule d'Euler:

$$\pi = \sqrt{6\sum_{i=1}^{\infty}\frac{1}{i^2}} \quad \text{on a} \quad \lim_{i\to\infty}\sqrt{6\sum_{i=1}^{\infty}\frac{1}{i^2}} = \pi$$

Pour nous, on cherche une valeur approchée de π avec un précision de 10^{-6}.
Pour cela, on calcul deux sommes *sommeCourant* et *sommePred*. On arrête le calcul, dés que

$|sommeCourant - sommePred| \leq 10^{-6}$.

Remarque:
plus qu'on augument la précision, plus qu'on s'approche de la valeur exacte de π.
http://www.geom.uiuc.edu/~huberty/math5337/groupe/digits.html

3.14159265358979323846264338327950288419716939937510582097494459
2307816406286208998628034825342117067982148086513282306647093844 6
0955058223172535940812848111745028410270193852110555964462294895 4
9303819644288109756659334461284756482 33

```
(*
* Lang: Free Pascal
* Time complexity: O(1/eps)
* space complexity: O(1)
*)
const
    eps = 1e-6;
var
    i, sommePred, sommeCourant: real;
begin
    i := 1;
    sommeCourant := 0;
    sommePred:= 0;
    repeat
        sommePred := sommeCourant;
        sommeCourant := sommeCourant + 1 / sqr(i);
        writeln(sqrt(6 * sommeCourant):0:10);
        i := i + 1;
    until abs(sqrt(6 * sommeCourant) - sqrt(6 * sommePred)) <= eps;
end.
```