Exercices sur les différents types de données

Exercice d'Arithmétique : Naturels premiers chinois

Il y a 2500 ans qu'en Chine fut émise la conjecture suivante :

Pour tout entier $n > 1$, n premier $\Leftrightarrow$ n divise $2^n - 2$

Nous appellerons *naturel premier chinois* ou encore *naturel pseudo-premier*, tout entier naturel $n > 1$ qui vérifie : n divise $2^n - 2$.

1) Ecrire un programme qui détermine tous les naturels pseudo-premiers jusqu'à 1000.

Fermat a montré en 1640 que pour tout naturel n :

n premier $\Rightarrow$ n divise $2^n - 2$

La réciproque est malheureusement fausse. (Serait-elle exacte que nous aurions là un excellent test pour reconnaître si un entier naturel est un nombre premier ou non)

2) Ecrire une fonction **[b]=premier(n)** qui détermine si un entier n donné en paramètre est un nombre premier ou non. Suivant l'entier n, la fonction renvoie alors le bouléen b contenant la valeur vrai ou faux.

3) Ecrire un programme qui écrit tous les entiers naturels premiers chinois non premiers, c'est-à-dire tous les contre-exemples de la relation établie par les chinois.

Exercice d'Approximations, le nombre π

1°) On rappelle que n ! est définie par $n ! = 1\times2\times3\times \ldots \times n$

Ecrire un programme Scilab permettant de calculer n ! où n est un nombre entier saisi au clavier.

2) Voici une méthode, parmi de nombreuses autres, de calcul d'une valeur approchée de π.

Le point de départ est la formule de Wallis (John Wallis 1616 – 1703) :

$$\frac{\pi}{2} = \sum_{k=0}^{+\infty} \frac{k\,!}{1\times3\times\ldots\times(2k+1)}$$

a) Modifier le programme précédent (du 1)) et le transformer en une fonction afin de calculer le terme $\dfrac{k\,!}{1\times3\times\ldots\times(2k+1)}$

b) Construire alors un autre fonction permettant de calculer les 10 premiers termes de la somme précédente et affichant la valeur approchée de π obtenue.

c) La formule de Wallis peut se transformer successivement en :

$$\begin{aligned}\frac{\pi}{2} &= 1 + \frac{1}{3} + \frac{1\times2}{3\times5} + \frac{1\times2\times3}{3\times5\times7} + \frac{1\times2\times3\times4}{3\times5\times7\times9} + \ldots \\ &= 1 + \frac{1}{3}\left(1 + \frac{2}{5}\left(1 + \frac{3}{7}\left(1 + \frac{4}{9}(1 + \ldots)\right)\right)\right)\end{aligned}$$

dont une approximation est, pour un entier k donné :

$$\frac{\pi}{2} \approx 1 + \frac{1}{3}\left(1 + \frac{2}{5}\left(1 + \frac{3}{7}\left(1 + \frac{4}{9}\left(\ldots\left(1 + \frac{k}{2k+1}\right)\right)\right)\right)\right)$$

1) Lorsque $k = 4$, simuler, sans utiliser la calculatrice, le calcul.

2) A partir d'un entier k donné au clavier, calculer la valeur approchée de π obtenue.

(Vous commencerez votre boucle de calcul par $1 + \dfrac{k}{2k+1}$)

Exercice sur les tableaux, le triangle de Pascal

1) Ecrire la fonction **[tab]=triangle_de_pascal(n)** permettant de construire un tableau de taille $n \times n$ représentant le triangle de Pascal comme sur l'exemple ci-dessous qui présente le résultat suite à l'appel de la fonction dans le cas particulier où $n = 5$. On rappelle la formule $\binom{n+1}{p+1} = \binom{n}{p} + \binom{n}{p+1}$

```
-->tab=triangle_de_pascal(5)
 tab  =

!  1.   0.   0.   0.   0. !
!  1.   1.   0.   0.   0. !
!  1.   2.   1.   0.   0. !
!  1.   3.   3.   1.   0. !
!  1.   4.   6.   4.   1. !
```

2) On souhaite retrouver un résultat connu sur la somme de chaque ligne d'un tel tableau.
Soit tab un tableau construit à partir de la fonction précédente (comme sur l'exemple).
Ecrire la fonction **[t]=somme(tab)** qui construit le tableau t contenant la somme des éléments de chaque ligne du tableau tab.
Que remarquez-vous sur le tableau ainsi construit. Expliquez-le !

Exercice sur les chaînes de caractères Le jeu de Robinson

Le jeu des consonnes :
Il s'agit de compléter la phrase suivante de manière à la rendre exacte : *Cette phrase contient .... consonnes"*
Une façon a priori stupide de résoudre le problème est d'écrire une autre phrase qui décrive cette première phrase. Comme elle contient 18 consonnes, nous obtenons :

Cette phrase contient dix-huit consonnes.

Faux, bien entendu, puisque le nombre de consonnes a été modifié. Nous la corrigeons donc en recommençant, nous obtenons successivement :

Cette phrase contient vingt-deux consonnes.
Cette phrase contient vingt-quatre consonnes.
Cette phrase contient vingt-cinq consonnes.
Cette phrase contient vingt-cinq consonnes.

Les deux dernières phrases sont identiques. Comme la seconde décrit la première, elles sont exactes !
Cette méthode est celle des approximations successives.

Le jeu à simuler est celui de Robinson (inventé dans les années 70 par l'américain Raphaël Robinson)
Le but est de remplir les blancs de la phrases suivante afin qu'elle devienne vraie :

Dans cette phrase, il y a : __ 0, __ 1, __ 2, __ 3, __ 4, __ 5, __ 6, __ 7, __ 8, __ 9.

Les premières étapes donnent :

Dans cette phrase, il y a : 1 0, 1 1, 1 2, 1 3, 1 4, 1 5, 1 6, 1 7, 1 8, 1 9.
Dans cette phrase, il y a : 1 0, 11 1, 1 2, 1 3, 1 4, 1 5, 1 6, 1 7, 1 8, 1 9.

Une variable de type tableau contiendra en chacun de ses position *i* le nombre de fois où apparaît le chiffre *i* dans la chaîne de caractère *c*.

1) Ecrire une fonction **[t]=compter(c);** permettant d'affecter aux éléments du tableau *t* le nombre des chiffres correspondants dans la chaîne *c*.
2) Dans le programme principal, la chaîne de caractère *c* sera initialisée à

c='Dans cette phrase, il y a 0, 1, 2, 3, 4, 5, 6, 7, 8, 9'

Le nombre de 0 sera ainsi donné en position 27, celui de 1 en position 32 (27 + 5), celui de 2 en position 37 (32 + 5).
Ecrire une fonction **[d]=Inserer_compte(c,t);** permettant de construire la chaîne *d* modification de la chaîne *c* et suivant le tableau *t* contenant le nombre d'occurrences des chiffres trouvés précédemment dans la chaîne *c*.
3) Construire le programme principal permettant de déterminer la chaîne exacte. Tant que la nouvelle chaîne est différente de la précédente, il faut recommencer.
Vous devrez afficher la chaîne exacte.

Correction exercices sur les différents types de données

Naturels premiers chinois

```
function pseudo(n)
for i=2:n do
   if modulo(2^i-2,i)==0 then disp(i); end
end

function [b]=premier(n)
b=%t
for i=2:floor(sqrt(n)) do
   if modulo(n,i)==0 then b=%f; end
end

function contre_exemples_chinois(n)
for i=1:n do
if (modulo(2^i-2,i)==0) & ~premier(i) then disp(i); end
end
```

Approximation de π par Wallis

```
n=input('entrez un entier positif');p=1;
for i=1:n do
  p=p*i;
end
disp(p)

k=input('entrez un entier positif');p1=1;p2=1;
for i=1:k do
  p1=p1*i;
  p2=p2*(2*i+1);
end
terme=p1/p2;disp(terme)

somme=0
for k=0:9 do
  p1=1;p2=1;
  for i=1:k do
    p1=p1*i;
    p2=p2*(2*i+1);
  end
  terme=p1/p2;somme=somme+terme;
end
disp('Une valeur approchée de pi par cette méthode est')
disp(2*somme)

k=input('entrez un entier positif');somme=1;
for i=k:-1:1 do
  somme=somme*i/(2*i+1)+1
end
disp('Une valeur approchée de pi par cette méthode est')
disp(2*somme)
```

Triangle de Pascal

```
function [tab]=triangle_de_pascal(n)
```

```
tab=zeros(n,n)
for i=1:n do
  tab(i,1)=1;
end
for i=2:n do
        for j=2:n do
                tab(i,j)=tab(i-1,j-1)+tab(i-1,j)
        end
end

function [t]=colonne_somme(tab)
n=sqrt(length(tab));t=zeros(n,1)
for i=1:n do
        for j=1:n do
                t(i)=t(i)+tab(i,j);
        end
end
//on devrait bien entendu retrouver le cardinal de l'ensemble des parties d'un ensemble, c'est-à-dire 2^n
```