| BCPST | Informatique : D.S. n°1 | Samedi 2 décembre |
|---|---|---|

**Vous devrez rendre 3 copies :** **Une copie pour les exercices 1 et 2**
**Une copie pour l'exercice 3**
**Une copie pour l'exercice 4**

Une partie du barème est donnée sur la syntaxe du langage Scilab et sur votre interprétation de l'algorithme nécessaire pour répondre à chaque problème. Vous devrez donc expliquer chaque programme par un texte en français.

**Exercice 1** (3 points)
Exécuter le programme suivant pas à pas en affichant le contenu de chacune des variables.
Dire ensuite ce qu'il réalise (pour la variable *p*)

```
-->p=1;i=1;
-->while i<5 do j=i+1;
-->p=p*j/i;
-->disp(p,' : p=',i,'étape')
-->i=i+1;
-->end
```

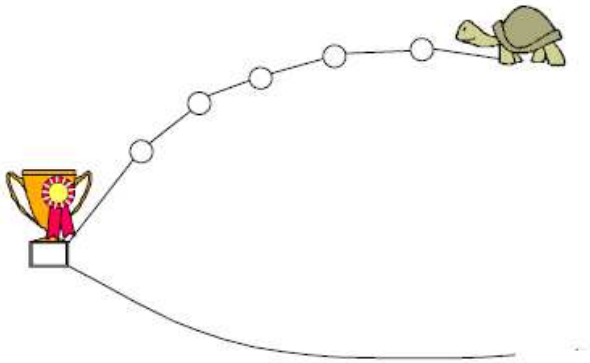
**Exercice 2** (5,5 points)
Le but de cet exercice est d'écrire un programme Scilab qui calcule la somme des chiffres d'un nombre entier strictement positif et recommence le calcul avec le résultat obtenu tant que celui-ci n'est pas compris entre 1 et 9. Après chaque calcul, la fonction affiche à l'écran la somme obtenue.
La fonction retournera (ou le programme affichera) le nombre entre 1 et 9 obtenu.
Exemple : Si le nombre initial est 123456, la fonction affichera
21 (car 1 + 2 + 3 + 4 + 5 + 6 = 21)
3 (car 2 + 1 = 3)
1) Ecrire une fonction (ou un programme) permettant de décomposer un nombre pour en distinguer les chiffres (vous supposerez que le nombre ne contient pas plus de 6 chiffres).
2) Utiliser la fonction ci-dessus (ou le programme ci-dessus) pour créer un programme Scilab permettant de répondre au problème donné.

**Exercice 3** (5,5 points)

1) Construire la fonction **function [s]=harmonique(n)** permettant de calculer $s = \sum_{k=1}^{n} \frac{1}{k}$ avec le paramètre d'entrée *n*.

2) On démontre que la suite $(u_n)$ définie pour tout $n \geqslant 1$ par $u_n = \sum_{k=1}^{n} \frac{1}{k} - \ln(n)$ est une suite décroissante, minorée par 0 et donc convergente.
Sa limite est appelée constante d'Euler *C* avec $C \approx 0{,}577\,215$.
Ecrire un programme Scilab utilisant éventuellement la fonction du 1), déterminant la valeur de *n* à partir de laquelle la valeur de $u_n$ diffère de moins d'un millième de *C*.
(Rappel : la fonction *ln* s'écrit *log* sous Scilab)

**Exercice 4** (6 points)
On lance un dé.
Si le 6 sort, le lièvre gagne.
Sinon, la tortue avance d'une case.
On continue jusqu'à ce qu'il y ait un gagnant.



Quelle est la situation la plus enviable, celle du lièvre ou celle de la tortue ?

1) Expliquer pourquoi la formule tapée sous Scilab : *floor*(*n*×rand()) + 1 permet d'obtenir un entier quelconque (et équiprobable) entre 1 et *n*.

2) Il s'agit dans cette question de simuler une seule partie en construisant une somme contenant les déplacements de la tortue : tant que le résultat n'est pas 6 ou que la tortue n'est pas arrivée, retirer un nombre quelconque entre 1 et 6.
Afficher alors le gagnant de la partie.

3) Modifier le programme précédent de manière à réaliser 100 simulations de parties.
Quelle est la situation la plus avantageuse, celle de la tortue ou celle du lièvre ?
Vous proposerez un affichage sous Scilab pour répondre à cette question.

# Correction du ds n°1

**Exercice 1**

```
-->p=1;i=1;                    p=1  i=1
-->while i<5 do j=i+1;         j=2          j=3          j=4          j=5
-->p=p*j/i;                    p=2          p=3          p=4          p=5
-->disp(p,' : p=',i,'étape')   étape 1 p=2  étape 2 p=3  étape 3 p=4  étape 4 p=5
-->i=i+1;                      i=2          i=3          i=4          i=5
-->end
```

La variable p prend ainsi les entiers successifs à partir de 1 jusqu'à 5.

**Exercice 2**

```
n=input('entrez un entier entre 1 et 999999')
a=floor(n/100000); n=n-a*100000;
b=floor(n/10000); n=n-b*10000;
c=floor(n/1000);n=n-c*1000;
d=floor(n/100);n=n-d*100;
e=floor(n/10);
f=n-10*e;
n=a+b+c+d+e+f;
disp(n)
```

```
while n>9 do
  a=floor(n/100000); n=n-a*100000;
  b=floor(n/10000); n=n-b*10000;
  c=floor(n/1000);n=n-c*1000;
  d=floor(n/100);n=n-d*100;
  e=floor(n/10);
  f=n-10*e;
  n=a+b+c+d+e+f;
  disp(n)
end
```

**Exercice 3**

1)
```
//création de la fonction
function [s]=harmonic(n)
s=1;
for i=2:n do
s=s+1/i;
end
```

2)
```
function []=programme()
n=1
c=0.577215
while abs(harmonic(n)-log(n)-c)>0.001 do
   n=n+1;
end
disp(n)
```

**Exercice 4**

**function []=une_partie()**
```
x=floor(6*rand())+1
s=0
while x<>6 & s<6 do
  s=s+1
  x=floor(6*rand())+1
end
if s<>6 then disp('le lièvre gagne')
       else disp('la tortue gagne')
end
```

**function simulation()**
```
l=0; t=0
for i=1:100 do
  x=floor(6*rand())+1;   s=0
  while x<>6 & s<6 do
    s=s+1; x=floor(6*rand())+1
  end
  if s<>6 then l=l+1, else t=t+1; end
end
if l>t then disp('la situation du lièvre est la plus
enviable')
elseif t>l then disp('la situation de la tortue est plus
enviable')
else disp('les deux situations sont enviables : il y a
égalité')
 end
```

| BCPST | Informatique : D.S. n°2 | Samedi 24 mars |
|---|---|---|

**Vous devrez rendre 3 copies :** **Une copie pour les exercices 1 et 2**
**Une copie pour l'exercice 3**
**Une copie pour l'exercice 4**

**Exercice 1** (3,5 points)
Le nombre *e* peut être défini comme la limite d'une série :

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \ldots + \frac{1}{n!} + \ldots$$

Il est connu que l'approximation $e = 1 + \frac{1}{1!} + \frac{1}{2!} + \ldots + \frac{1}{n!}$ diffère de *e* d'au plus de deux fois le terme suivant dans la somme infinie, c'est-à-dire d'au plus $2\frac{1}{(n+1)!}$.

1) Construire la fonction **[p]=factorielle(n)** permettant de calculer la factorielle de l'entier *n*.
2) En utilisant la fonction précédente, écrire une programme calculant *e* avec une approximation dont l'erreur est inférieure à une constante c donnée arbitrairement.

## *Exercice 2 (3,5 points)*

1) Ecrire une fonction **[bool]=janus(n)** permettant de déterminer si l'entier *n* donné en paramètre est un nombre de Janus. Un nombre ou un mot de Janus peut se lire indifféremment dans les 2 sens, comme 13431 ou ressasser.
2) Ecrire un programme Scilab qui conserve dans un tableau tous les nombres de 1 à *N* , entier donné au clavier, dont le carré est un nombre de Janus.
Vous afficherez le tableau obtenu.

## *Exercice 3 Première loi de Mendel (6 points)*

Ayant des individus homozygotes *AA* et *aa* dans la lignée parentale, la première génération de fille F1 n'aura que des individus hétérozygotes *Aa* de phénotype *A*. En les croisant entre eux, on attend la proportion "3 pour 1" entre le phénotype *A* (correspondant aux génotypes *AA*, *Aa* et *aA*) et le phénotype *a* (correspondant au génotype *aa*).
Cependant, si vous comptez les phénotypes de 100 individus de la F2, vous n'aurez que très rarement les bonnes proportions (c'est-à-dire 25 phénotypes *a* et 75 phénotypes *A*). Vous allez utiliser Scilab pour déterminer la distribution du nombre de phénotypes *a* dans ce type d'expérience.

1) Ecrire une fonction **[gen]=mendel()** qui génère aléatoirement un génotype *AA, Aa, aA* ou *aa*, la variable gen attendue sera de type chaîne de caractères.

2) Utiliser la fonction gen précédente pour construire la fonction **[n]=simulation_100()** qui permet de compter le nombre n de phénotypes *a* à partir de 100 simulations réalisées par la fonction gen.

3) Construire la fonction **[tab]=simulation()** permettant de construire le tableau tab de taille 1000. Ce tableau doit contenir les résultats de 1000 simulations données par la fonction simulation_100 précédente.

4) Construire une fonction **[moy]=moyenne(t)** permettant de déterminer la moyenne moy d'un tableau t.
Appliquez alors les fonctions précédentes dans un programme pour déterminer la distribution sur une simulation donnée par la fonction du 3).

**Exercice 4 : Jeu de Nim** (6 points)
Le jeu se joue à deux : toi (l'élève) contre l'ordinateur (le prof). C'est chacun à son tour de jouer.
Le tapis comporte une seule rangée de 21 allumettes (on prendra la variable *n* pour le nombre d'allumettes).
Lorsque c'est à ton tour de jouer, tu dois enlever une, deux ou trois allumettes.
Celui qui retire la dernière allumette perd la partie.

Il s'agit de te laisser jouer le premier et de te faire perdre à tous les coups en construisant la fonction []=Nim() sans paramètres (qu'il faut écrire en Scilab), dont l'algorithme est le suivant :

tant que la partie n'est pas finie :
- afficher le message "Il reste xxxx allumettes, vous devez en retirer entre 1 et 3" où xxxx représente le nombre d'allumettes restantes.
- demander à l'utilisateur un nombre entre 1 et 3. Redemander ce nombre s'il ne se trouve pas entre 1 et 3, et mettre à jour le nombre d'allumettes disponibles.
- l'ordinateur doit enlever à son tour des allumettes.
  La stratégie qu'il doit adopter est d'enlever le complément à 4 d'allumettes (C'est à dire : si vous enlevez 1 allumette, l'ordinateur en enlève 3, si vous en enlevez 2 l'ordinateur 2 et si vous en enlevez 3 l'ordinateur en enlève 1).
- S'il ne reste plus d'allumette après votre tour de jeu, afficher : « L'ordinateur a gagné ».

Si c'est après le tour de l'ordinateur (???), afficher : « L'ordinateur a perdu ».


Le roi Janus l'homme aux deux visages

Caricature représentant Louis XVI sous la forme de Janus aux deux visages. Qui d'un coté prête serment aux représentants de la Nation

**"Je soutiendrais la Constitution"**
et de l'autre affirme aux représentants de l'église
*"Je détruirai la Constitution ".*

# Correction DS n°2

```
function [p]=factorielle(n)
p=1;
for i=1:n do p=p*i;end

function []=pg1()
c=0.001
//recherche de l'indice nécessaire
n=0
while 2/factorielle(n+1)>c do n=n+1;end
//calcul de la somme
e=1
for i=1:n do e=e+1/factorielle(i); end
disp(e)

function [bool]=janus(n)
c=string(n); l=length(c)
bool=%t;
for i=1:floor(l/2) do
  if part(c,i)<>part(c,l-i+1) then bool=%f; end
end

function []=pg2()
j=1; t=[]
N=input('Jusqu"à quel entier voulez-vous travailler ?')
for i=1:N do
  if janus(i^2)==%t then t(j)=i;j=j+1; end
end
disp(t)

function [gen]=mendel()
x=floor(4*rand())
select x
case 0 then gen='AA'
case 1 then gen='Aa'
case 2 then gen='aA'
case 3 then gen='aa'
end

function [n]=simulation_100()
n=0
for i=1:100 do
  if mendel()=='aa' then n=n+1;   end
end

function [tab]=simulation()
tab=zeros(1,1000)
for i=1:1000 do tab(i)=simulation_100(); end

function [moy]=moyenne(t)
s=0
for i=1:1000 do s=s+t(i); end
moy=s/1000
```

```
function []=pg3()
t=simulation(); m=moyenne(t)
disp('La distribution est, en moyenne ')
disp(string(floor(m*10)/10)+'% des cas pour aa')
//remarque : deux applications de cette fonction ont donné 24,9% et 25,1%

function []=Nim();
x=21;
while x>=1 do
        disp('il reste '+string(x)+' allumettes, combien souhaitez-vous en retirer ?')
        n=input('Donnez un nombre entre 1 et 3');
        while n<1 | n>3 do
                n=input('Vous devez donner un nombre entre 1 et 3 !');
        end
        x=x-n;//on retire les allumettes du joueur
        if x==0 then disp('vous avez perdu');end
        x=x-(4-n);//l'ordinateur fait le complément à 4
        if x==0 then disp('l''ordinateur a perdu');end
end
```

| BCPST | Informatique : D.S. n°3 | Samedi 9 juin |
|---|---|---|

**Vous devrez rendre 3 copies : Une copie pour l'exercice 1**
**Une copie pour l'exercice 2**
**Une copie pour l'exercice 3**

**Exercice 1 Un générateur de Charabia latin** (7 points)
Nous nous proposons de réaliser un générateur de charabia latin (jeu utilisé par les petits enfants de langue anglaise), c'est-à-dire un programme qui modifie un mot du français en un mot d'un charabia latin. Cette transformation s'effectue en plaçant la première lettre du mot à la fin, et en y ajoutant la lettre "a". Ainsi, le mot "tortue" devient "ortueta", "Scilab" devient "cilaba", et ainsi de suite.
Ecrivons maintenant un programme Scilab qui lira une phrase en français et écrira son équivalent en charabia latin. Pour des raisons de simplicité, nous ne tiendrons pas compte du problème des lettres majuscules et des signes de ponctuation.

L'écriture du programme traduisant le fonctionnement de ce jeu sera facilitée par la construction des modules suivants :

1) Ecrire la fonction **[ch]=Entrez_Texte()** permettant d'entrer le texte en français représenté par la chaîne de caractère **ch** (vérifiez que le texte comporte moins de 80 caractères).
2) Ecrire la fonction **[n]=Nombre_mots()** permettant de déterminer le nombre **n** de mots de la phrase **ch** en utilisant le nombre d'espaces.
3) Ecrire la fonction **[m]=recherche_mot(ch,n)** permettant d'affecter à la chaîne **m** le **n**$^{\text{ième}}$ mot de la phrase **ch**.
4) Ecrire la fonction **[ma]=Transf_Franc_Charabia(mf)** permettant de transformer le mot français de la chaîne **mf** en son équivalent en charabia latin dans la chaîne **ma**.
5) Ecrire le programme complet permettant de transformer une phrase complète en son équivalent en charabia latin.

**Exercice 2** (7 points)
**Pour chacune des questions suivantes, vous donnerez uniquement les lignes de programme Scilab permettant de donner la réponse, avec d'éventuels commentaires supplémentaires permettant de les expliquer. Aucun calcul "à la main" n'est demandé.**

1) Un chocolatier fabrique trois sortes de chocolat avec du cacao, du lait, du sucre et du beurre. Le tableau suivant donne les quantités d'unités nécessaires à la fabrication d'une unité de chaque sorte de chocolat.

| **Type de chocolat** | **Cacao** | **Lait** | **Sucre** | **Beurre** |
|---|---|---|---|---|
| ***Asia*** | 6 | 4 | 4 | 0 |
| *América* | 7 | 5 | 3 | 2 |
| *Africa* | 8 | 2 | 2 | 4 |

a) Représenter les données par une matrice $M$ de dimension $4\times3$.
b) Il reçoit une commande de 5 unités de chocolat *Asia*, 6 unités de chocolat *América* et 9 unités de chocolat *Africa*.
- Calculer les quantités nécessaires pour chaque produit.
- Les prix respectifs de chaque composant, en euros par unités, sont respectivement 4, 3, 5 et 6.
  Calculer le prix total pour cette commande.

2) Modélisation d'un échange entre deux milieux

Deux récipients *A* et *B* sont séparés par une membrane perméable dans les deux sens. On place dans les récipients *A* et *B* deux solutions contenant respectivement $a_0$ molécules (dans *A*) et $b_0$ molécules (dans *B*). On suppose que, toutes les heures, 20% des molécules passent de *A* dans *B* et 10% des molécules passent de *B* dans *A*. On note $a_n$ et $b_n$ les nombres respectifs de molécules présentes dans *A* et *B* au bout de *n* heures.

a) Montrer que $\begin{cases} a_{n+1} = 0{,}8a_n + 0{,}1b_n \\ b_{n+1} = 0{,}2a_n + 0{,}9b_n \end{cases}$ et donner l'interprétation matricielle de ce système en considérant la matrice colonne $p_n = \begin{pmatrix} a_n \\ b_n \end{pmatrix}$.

Les deux récipients n'ayant d'échanges qu'entre eux.

b) Sachant que si $a_0 = 150$ et $b_0 = 20$ (unités), quelles instructions écrire pour connaître les quantités de molécules après 10 heures ?

Quelle méthode appliqueriez vous pour connaître la répartition limite, si elle existe, entre les deux milieux ?

c) Quels sont les dosages initiaux nécessaires pour obtenir après 1 heure, une répartition égale à $a_1 = 130$ et $b_1 = 40$ (unités). Ecrire l'instruction Scilab permettant d'expliciter le résultat.

**Exercice 3 Le suicide des Zélotes** (6 points)

L'exercice s'inspire du suicide des zélotes à Masada en avril 74. La formulation ci-après est tirée du livre, *Concrete Mathematics* de Graham, Knuth et Patashnik publié chez Addison Wesley en 1989 :

> *Préférant le suicide à la capture, les rebelles juifs cachés dans une grotte décidèrent de former un cercle et, en le parcourant, de tuer chaque troisième personne restante jusqu'à ce qu'il n'y ait plus personne. Mais Flavius Josèphe et un co-conspirateur inconnu ne voulaient pas de ce suicide insensé ; ainsi il détermina rapidement où il devait se placer (ainsi que son ami) sur le cercle vicieux.*

Le but de cet exercice est de déterminer, grâce à Scilab et à partir d'une origine sur ce cercle (la première personne tuée), les deux dernières personnes concernées (Flavius Josèphe et son ami).

Vous supposerez pour cet exercice qu'il y a *n* personnes formant un cercle.

Ces *n* personnes seront modélisées par un tableau de taille *n* contenant la valeur 0 si l'individu associé est sauf et 0 si l'individu est tué.

1) Construire la fonction [tab]=ordre_des_tues(n) permettant :

- de construire le tableau t des individus, considérés comme tous saufs au début.
- de construire un tableau tab de même taille et contenant également que des zéros. Ce tableau va permettre de contenir tous les indices de tous les individus successifs tués.
- de faire mourir le premier individu (celui d'indice 1) et de le noter dans le tableau tab.
- tant que le tableau t contient encore des zéros :
  - ajouter 3 à l'indice de la personne
  - tant que cela donne un indice d'une personne tuée, aller à l'indice suivant
  - si l'indice dépasse n, revenir à 1 (et même 2, voire plus)
  - tuer la première personne vivante rencontrée et recommencer les trois étapes précédentes.

2) Construire la fonction []=affichage(tab) qui, après avoir déterminé la longueur de tab permet un affichage du type :

"Flavius Josèphe doit se trouver en ?ème position pour être sauf."

"L'ami de Flavius doit se trouver en ?ème position pour être sauf."

**Correction du DS n°3**

Exercice 3

```
function [tab]=ordre_des_tues(n)
tab=zeros(1,n)
t=tab
i=1;s=1;tab(1)=1;t(1)=1;

while s<n do
  i=i+3;
  while i <=n & t(i)==1 do i=i+1; end
  if i>=n+1 then i=2;end
  while t(i)==1 & i <=n do i=i+1; end

  t(i)=1;s=s+1;tab(s)=i;
end

function []=affichage(tab)
  n=length(tab)
  disp("Flavius Josèphe doit se mettre en "+string(tab(n))+"ème position.")
  disp("L"ami de Flavius Josèphe doit se mettre en "+string(tab(n-1))+"ème position.")
```