Algorithmes et instructions

1 Introduction aux algorithmes

1.1 Pseudo-langage

Exercice 1. Ca permute, ça permute !.
L'objectif de ces questions est de rédiger un algorithme permettant d'échanger deux variables.

1) Définissez les paramètres de l'algorithme.

Correction. Les paramètres sont les entiers `a` et `b` que l'on doit permuter.

2) Cet algorithme produit-il des sorties ? Si oui, explicitez-les.

Correction. La sortie de cet algorithme consiste à donner les variables `a` et `b` dont la valeur a été échangée.

3) Cet algorithme nécessite l'usage d'une variable temporaire. Où est-elle définie ?

Correction. Les variables nécessaires au bon déroulement d'un algorithme, au contraire des paramètres, doivent être définies à l'intérieur de l'algorithme.

4) Rédigez enfin cet algorithme.

Correction.
Algorithme 1 – Échange de variables
Entrée. `Entiers a,b`
Sortie. `les entiers a et b permutés.`

```
Entiers tmp
tmp ← a
a ← b
b ← tmp
Renvoyer a,b
```

1.2 Traduction en langage C

2 Instructions conditionnelles

2.1 Instruction `if/else`

Exercice 2. Un peu de pratique.
Considérons l'algorithme suivant et sa traduction en langage C juste à droite.

Correction

Algorithme 2 – Qui suis-je?
Entrée. Entiers a,b,c
Sortie. Affichage de valeurs

```
Entiers M, m, n
Si a > b alors
    M ← a
    m ← b
Sinon
    M ← b
    m ← a
Fin Si
Si c > M alors
    M ← c
Fin Si
Si c < m alors
    m ← c
Fin Si
n ← a+b+c-M-m
Afficher m, n, M
```

```c
int a=3, b=5, c=1;
int M, m, n;
if (a>b)
{
  M = a;
  m = b;
}
else
{
  M = b;
  m = a;
}
if (c>M)
  M = c;
if (c<m)
  m = c;
n = a+b+c-M-m;
printf("m=%d, n=%d, M=%d\n",
 ↪  m,n,M);
```

1) Qu'affiche cet algorithme sur les valeurs d'entrée a=3, b=5 et c=1 ?

Correction. Il affiche 1, 3, 5.

2) Qu'affiche cet algorithme sur les valeurs d'entrée a=-2, b=0 et c=4 ?

Correction. Il affiche -2, 0, 4

3) Qu'affiche cet algorithme dans le cas général ?

Correction. Il affiche la valeur minimale m, la valeur médiane n et la valeur maximale M.

4) Copiez le code C, compilez-le, exécutez-le et vérifiez vos réponses.

Exercice 3. Un peu de pratique.
Soit a, b et c trois variables entières initialisées à a=0, b=2 et c=-1. Évaluez les expressions suivantes puis vérifiez vos résultats en rédigeant un programme C calculant et affichant la valeur de ces expressions.

1) `a < (b+c) = 1`
2) `b = a = 0`
3) `(c*b) > a = 0`
4) `(b>a) && (b>c) = 1`
5) `(a>b) || (a>c) = 1`
6) `c <= a <= b = 1`
7) `(c<a) && (a<b) = 1`
8) `(a!=0) && (b!=0) = 0`
9) `(a!=0) || (b!=0) = 1`

Correction.

10) `a < (b+c) = 1`
11) `b = a = 0`
12) `(c*b) > a = 0`
13) `(b>a) && (b>c) = 1`
14) `(a>b) || (a>c) = 1`
15) `c <= a <= b = 1`
16) `(c<a) && (a<b) = 1`
17) `(a!=0) && (b!=0) = 0`
18) `(a!=0) || (b!=0) = 1`

Exercice 4. Un peu de pratique (Moodle).
Reprenez le code du rendu de monnaie que vous avez réalisé lors du chapitre précédent. Ajoutez un test à votre code afin de vérifier que l'utilisateur donne bien une somme

Correction

supérieure ou égale à celle qui lui est demandée. Si ce n'est pas le cas, affichez un message d'erreur.

Correction. Voir la correction sur Moodle

Exercice 5. Attention aux zéros !.

On définit un ensemble de variables de la façon suivante.

```
float x=0, y=0, z;
int a=0;
```

Compilez, exécutez et expliquez le résultat des codes suivants.

```
if ((x!=0) && ((1/x)>2))
  printf ("C'est plus grand\n");
if (((1/x)>2) && (x!=0))
  printf ("Vraiment plus grand\n");
```

```
if (!(x=y))
  z = 1/(x-y);
else
  z = 0;
```

```
if (a!=0);
  printf ("a ne vaut pas 0\n");
```

```
if (a=0)
  printf ("a vaut 0\n");
```

Correction. Le code en haut à gauche réalise les tests dans un ordre différent. Le premier `if` teste d'abord si `x!=0` puis, seulement si `x` est bien différent de 0, si `(1/x)>2`. Il évite donc de diviser 1 par 0. Le second `if` tombe dans le piège de la division par 0.

En haut à droite, l'expression `!(x=y)` signifie en réalité « non `y` ». En effet, on réalise d'abord l'affectation `x=y` qui renvoie la valeur de `y`, puis la négation est réalisée. Si on voulait tester si `x` et `y` était différent, il aurait fallu écrire `x!=y`

En bas à gauche, le texte s'affiche alors que `a=0` à cause du point virgule oublié après le `if`. Dans ce cas, l'instruction vide est réalisée si `a!=0` et dans tous les cas, le message s'affiche ensuite.

Enfin, en bas à droite, le texte ne s'affiche pas alors que `a=0`. Comme cette expression est réalisée dans le `if`, elle est évaluée par le `if`. L'expression `a=0` retournant 0, on ne rentre pas dans le `if`, donc le message n'est pas affiché.

Exercice 6. Quel âge as-tu ?.

Considérons l'extrait de code suivant.

```
int test = 0;
printf ("Quel age avez vous?\n");
scanf ("%d", &test);

if (test<=3)
  printf ("On n'est pas des idiots!\n");
else
{
  if (test<=32)
    printf ("Jeunot!\n");
```

Correction

```
  else
  {
    if (test <= 50)
      printf ("Ca passe encore\n");
    else
      printf ("Sorry\n");
  }
}
```

Répondez aux questions suivantes sans compiler ni exécuter ce code.

1) Qu'affiche ce code si la valeur saisie par l'utilisateur est 15 ?

Correction. Le programme affiche « Jeunot ! »

2) Qu'affiche ce code si la valeur saisie par l'utilisateur est 32 ?

Correction. Le programme affiche toujours « Jeunot ! »

3) Déterminez la tranche d'âge concernée par chaque message affiché.

Correction. Entre 0 et 3 ans, le programme affiche « On n'est pas des idiots ! ». Entre 4 et 32 ans, le programme affiche « Jeunot ! ». Entre 33 et 50 ans, le programme affiche « Ca passe encore ». Au delà de 50 ans, le programme affiche « Sorry ».

4) Reformulez chacune des conditions à l'aide du ET logique et/ou du OU logique.

Correction.

```
if (test<=3)
  printf ("On n'est pas des idiots!\n");
if (4 < test && test<=32)
  printf ("Jeunot!\n");
if (33 < test && test<=50)
  printf ("Ca passe encore!\n");
if (test>50)
  printf ("Sorry\n");
```

Exercice 7. Racines d'un polynôme (Moodle).

Rédigez un algorithme permettant de résoudre l'équation $ax^2 + bx + c = 0$, avec a, b et c dans $\mathbb{R}$. Pour cela, l'algorithme doit prendre en entrée les trois coefficients de l'équation et doit afficher le résultat final. Il retournera le nombre de solutions de l'équation.

Implémentez ensuite votre algorithme en C. Les coefficients seront récupérés à l'aide de `scanf`.

Correction

Correction.

Algorithme 3 – Équation du second degré

Entrée. Réels a,b,c

Sortie. le nombre de solutions de l'équation $ax^2+bx+c = 0$

```
Réel D ← b² – 4ac
Réels s1, s2
Si D < 0 alors
    Afficher "Pas de solution"
    Renvoyer 0
Fin Si
Si D = 0 alors
    s1 ← -b/2a
    Afficher "Une solution: <s1>"
    Renvoyer 1
Fin Si
Si D > 0 alors
    s1 ← a/b (-b-√D)/(2a)
    s2 ← (-b+√D)/(2a)
    Afficher "Deux solutions: <s1> et <s2>"
    Renvoyer 2
Fin Si
```

Voir le code corrigé sur Moodle.

2.2 Instruction `switch/case`

Exercice 8. Un peu de pratique.

Considérons l'extrait de code suivant.

```
int nombre;
printf ("Entrez un nombre entier entre 1 et 3\n");
scanf ("%d", &nombre);
switch (nombre)
{
  case 1:
  printf ("Vous avez saisi 1\n");
  break;
  case 2:
  printf ("Vous avez saisi 2\n");
  break;
  case 3:
  printf ("Vous avez saisi 3\n");
  break;
  default:
  printf ("Votre nombre n'est pas entre 1 et 3\n");
}
```

1) Copiez ce code dans un `main()` et compilez-le.

Correction

2) Qu'affiche ce code avec la valeur d'entrée 1 ?

Correction. Ce code affiche « Vous avez saisi 1 ».

3) Qu'affiche ce code avec la valeur d'entrée 2 ?

Correction. Ce code affiche « Vous avez saisi 2 ».

4) Qu'affiche ce code avec la valeur d'entrée 37 ?

Correction. Ce code affiche « Votre nombre n'est pas entre 1 et 3 ».

5) Supprimez le second `break` et relancez votre code avec les valeurs 1, 2 et 37.

Correction. Pour la valeur 1, il affiche la même chose qu'avant.

Pour la valeur 2, il affiche « Vous avez saisi 2 » et « Vous avez saisi 3 ».

Pour la valeur 37, il affiche toujours « Votre nombre n'est pas entre 1 et 3 ».

6) En déduire l'utilité du `break`.

Correction. Le `break` permet d'arrêter la série d'instructions à réaliser lorsqu'on est entré dans un `case`.

Exercice 9. Poids de Linux (Moodle).

Considérons l'extrait de code suivant.

```
int poids;
printf ("Quel est le poids de Linux?\n");
scanf ("%d", &poids);
switch (poids/10)
{
  case 0:
    printf ("C'est pas un lapin!\n");
  break;
  case 1:
    printf ("Anorexique\n");
  break;
  case 2:
    printf ("Maigre\n");
  break;
  case 3:
    printf ("Parfait!\n");
  break;
  case 4:
    printf ("Gros!\n");
  break;
  default:
    printf ("C'est pas un ours!\n");
}
```

Précisons que si les variables `a` et `b` sont des entiers, la division de `a` par `b` produit un résultat entier. On a alors `a/b` = $\lfloor \frac{a}{b} \rfloor$. Répondez aux trois premières questions de cet exercices avant de compiler ce code.

Correction

1) Qu'affiche ce code avec la valeur d'entrée 37 ?

Correction. Le programme affiche « Parfait ».

2) Qu'affiche ce code avec la valeur d'entrée 42 ?

Correction. Le programme affiche « Gros ! ».

3) Qu'affiche ce code avec la valeur d'entrée 55 ?

Correction. Le programme affiche « C'est pas un ours ! ».

4) Copiez ce code dans un `main()`, compilez-le et vérifiez vos réponses aux questions précédentes.

5) Transformez la structure `switch`/`case` de ce code en `if`/`else`.

Correction.

```
if (poids < 10)
  printf ("C'est pas un lapin!\n");
if (10 <= poids && poids < 20)
  printf ("Anorexique!\n");
if (20 <= poids && poids < 30)
  printf ("Maigre!\n");
if (30 <= poids && poids < 40)
  printf ("Parfait!\n");
if (40 <= poids && poids < 50)
  printf ("Gros!\n");
if (poids >= 50)
  printf ("C'est pas un ours!\n");
```

Exercice 10. Matières toxiques ! (Moodle).

Le code suivant est sale, vous allez devoir nettoyer tout ça.

```
if ( a == 0 )
  printf( "Ca vaut 0\n" );
if ( a == 1 )
  printf( "Ca vaut 1\n" );
if ( a == 2 )
  printf( "%d\n", a );
if ( a == 3 )
  printf( "%d\n", a );
if ( a == 5 )
  printf( "%d\n", a );
if ( a == 4 )
  printf( "%d\n", a );
if ( (a!=0) && (a!=1) && (a!=2) && (a!=3) && (a!=4) && (a!=5) )
  printf( "Ca vaut un truc\n" );
```

1) Dans un premier temps, réécrivez-le en conservant la structure `if`/`else`.

Correction

Correction.

```
if ((a==0) || (a==1))
  printf( "Ca vaut %d\n", a );
else if ((2<=a) && (a<=5))
  printf( "%d\n", a );
else
  printf( "Ca vaut un truc\n" );
```

2) Réécrivez-le maintenant à l'aide de la structure `switch`/`case`.

Correction.

```
switch (a)
{
  case 0:
  case 1:
    printf( "Ca vaut %d\n", a );
  break;
  case 2:
  case 3:
  case 4:
  case 5:
    printf( "%d\n", a );
  default:
    printf( "Ca vaut un truc\n" );
}
```

3 Instructions de boucle

3.1 Instructions `while`, `do/while`

Exercice 11. Un peu de pratique.

Considérons le code présenté dans le cours, section 3.1.

1) Copiez ce code dans un `main` et compilez-le.
2) Qu'affiche ce code avec la valeur d'entrée 3 ?

Correction. Il affiche 3 fois les éléments de la première boucle, et 3 fois les éléments de la seconde boucle.

3) Qu'affiche ce code avec la valeur d'entrée 2 ?

Correction. Il affiche 2 fois les éléments de la première boucle, et 3 fois les éléments de la seconde boucle.

4) Qu'affiche ce code avec la valeur d'entrée 0 ?

Correction. Il n'affiche aucun élément de la première boucle, et 1 fois les éléments de la seconde boucle.

5) Expliquez la différence entre un `while` et un `do`/`while`.

Correction. Avec un `do/while`, on est sûr de réaliser les instructions de la boucle au moins une fois, pas avec un `while`. En effet, le `do/while` réalise les instructions puis teste la condition d'arrêt, à l'inverse du `while` qui teste d'abord la condition d'arrêt.

6) Supprimez l'instruction `croquettes--;` de la première boucle. Que se passe-t-il ? Utilisez la combinaisons de touches `CTRL-C` ou `CTRL-Z` si besoin.

Correction. Le programme tourne en boucle car la condition d'arrêt reste toujours vraie.

Exercice 12. Un peu de pratique.

On se base sur le code présenté dans la section 3.1 du cours.

1) Traduisez l'ensemble du code en langage algorithmique.
2) Modifiez le code pour que Linux mange les croquettes 2 par 2 dans chaque boucle.
3) Ajoutez un test permettant de vérifier que l'utilisateur a saisi une valeur positive.

Exercice 13. Je pense à . . . (Moodle).

Dans cet exercice, le but pour l'utilisateur est de trouver à quel nombre pense son voisin. Le voisin saisit son nombre mystère entre 1 et 5 et l'utilisateur a trois tentatives pour le trouver.

1) Rédigez un code permettant au voisin de saisir un nombre mystère.
2) Ajoutez un test vérifiant que le nombre saisi est bien entre 1 et 5. Sinon, le voisin est invité à saisir à nouveau le nombre.
3) Rédigez la ligne de code `system ("clear");` qui vous permet de nettoyer la console.
4) Dans une boucle `while`, Affichez le message `A quel nombre je pense?` puis récupérez la saisie de l'utilisateur.
5) Si c'est le nombre généré par l'ordinateur, affichez `WIN!` puis quittez le programme.
6) Si les trois tentatives ont été utilisées, affichez le nombre recherché et quittez le programme.

Correction. Voir la correction dans le fichier *Exo_13_je_pense_a.c*

Exercice 14. Souviens-toi l'entier dernier (Moodle).

L'objectif de cet exercice est de récupérer une série d'entiers positifs et retourner le plus grand et le plus petit élément. Bien évidemment, il vous est interdit d'utiliser plus de 5 variables ou des tableaux.

1) Demandez à l'utilisateur de saisir 10 entiers positifs et affichez-lui le plus grand.
2) Ajoutez l'affichage du plus petit.
3) Modifiez votre code pour que l'utilisateur puisse saisir autant d'entiers qu'il veut. La fin de la série sera repérée via la saisie d'un entier négatif.

Correction. Voir le fichier *Exo_14_max_min.c*

3.2 Instruction `for`

Exercice 15. Un peu de pratique.

Considérons le code ci-dessous.

Correction

```c
int i, j=2;                 // Variables de boucle

for (i=0; i<5; i++)         // Boucle 1
  printf ("Boucle 1: %d croquettes mangées.\n", i);

for (i=0; i<5; i++, j+=2)   // Boucle 2
  printf ("Boucle 2: %d nouvelles croquettes mangées.\n", j);

for (   ;  i>0;    )        // Boucle 3
  printf ("Boucle 3: %d croquettes restantes.\n", --i);
```

1) Copiez ce code dans un `main()` et compilez-le.
2) Combien de fois la première boucle est-elle exécutée ?

Correction. La boucle 1 s'affiche 5 fois.

3) Combien de fois la seconde boucle est-elle exécutée ?

Correction. La boucle 2 s'affiche 5 fois, mais compte de 2 en 2.

4) Combien de fois la troisième boucle est-elle exécutée ?

Correction. La boucle 3 s'affiche aussi 5 fois.

Étudions maintenant plus en détails la première boucle.

5) Remplacez dans le premier `for` l'expression `i=0` par `i=2` et exécutez à nouveau votre code.

Correction. La boucle 1 ne s'affiche plus que 3 fois et commence à 2.

6) Remplacez dans le premier `for` l'expression `i=0` par `i=4` et déduisez-en l'utilité de cette première expression.

Correction. La boucle 1 ne s'affiche plus que 1 fois et commence à 4. Cette première instruction dans un `for` permet d'initialiser le compteur de boucle.

7) Remettez l'instruction `i=0` et remplacez `i<5` par `i<7`. Déduisez-en l'utilité de cette seconde expression.

Correction. Cette seconde instruction est la condition d'arrêt de la boucle. La boucle tourne tant que cette expression est vraie.

8) Supprimez l'expression `i++` et exécutez votre code à nouveau. Que se passe-t-il ?

Correction. Le programme tourne en boucle.

9) Remplacez `i++` par `i+=2` et déduisez-en l'utilité de cette troisième expression.

Correction. La boucle tourne sur 2 fois moins de tours. Cette troisième expression permet d'incrémenter le compteur.

Exercice 16. Un peu de pratique.

Utilisons à nouveau l'extrait de code de la section 3.1.

1) Traduisez l'ensemble du code en langage algorithmique en utilisant exclusivement des boucle `Pour`.
2) Modifiez les boucles `while` du code pour les traduire en boucle `for`.

Correction

Exercice 17. Un peu de pratique (Moodle).

Les questions de cet exercice vont vous permettre de calculer la puissance et la factorielle d'entiers. On cherche donc à créer un programme permettant de calculer d'une part a^b et d'autre part $a!$ avec a et $b \in \mathbb{N}$. Rappelons que

$$a^b = \prod_{i=1}^{b} a \text{ et } a! = \prod_{i=1}^{a} i.$$

1) Sachant qu'une boucle est nécessaire dans les deux cas, rédigez un algorithme permettant de calculer a^b et un permettant de calculer $a!$.
2) Rédigez les programmes C répondant au problème.

Exercice 18. Que fais-je ?.

Sans compiler, ou exécuter ce code, qu'affiche le programme suivant ?

```
int res,n;
for(res=1,n=0;n<8;n=n+1)
{
  res=2*res;
  printf("res=%d\n",res);
}
```

Pour vous aider, vous pouvez compléter le tableau suivant donnant la valeur des variables `n` et `res` au cours du temps.

n	0	1	2	3	4	5	6	7	8
res	1	2	4	8	16	32	64	128	256

Incluez ce code dans un `main`, compilez-le et exécutez-le afin de vérifier votre travail.

Correction. Ce code affiche la suite des 2^n pour n compris entre 0 et 8.

Exercice 19. La boulette.

L'extrait de code suivant comporte quelques erreurs et/ou avertissements. Avant de le compiler, trouvez et corrigez ces erreurs, puis vérifiez votre correction en compilant ce code.

```
int i;j;
scanf("%d %d",i,j);
for(i=0,j=1,i>3;i=i+1,j=j+2)
printf("i=%d, j=%d\n,i,j)
if (i=j)
i=2*j
else
printf("%d\n",j);
printf("\nAu revoir\n");
```

Correction

Correction. La liste d'erreurs et/ou avertissements est la suivante :

- Ligne 1 : point virgule entre `i` et `j` qui aurait dû être une virgule ;
- Ligne 2 : oubli des `&` avant `i` et `j` ;
- Ligne 3 : la virgule entre `j=1` et `i>3` aurait dû être un point virgule ;
- Ligne 3 : la condition d'arrêt doit être `i<3` ;
- Ligne 4 : les guillemets devraient se fermer après le `\n` et il manque le point-virgule à la fin de la ligne ;
- Ligne 5 : le égal doit être un double égal ;
- ligne 6 : il manque le point-virgule à la fin de la ligne ;

Exercice 20. Un peu d'harmonie dans ce monde (Moodle).

Cet exercice consiste à calculer le n-ième terme de la série harmonique définie pour tout entier $n > 0$ par

$$u_n = \sum_{i=1}^{n} \frac{1}{i}.$$

Votre programme doit donc :

1) demander à l'utilisateur `Rang de la série harmonique?`, et récupérer la valeur saisie dans une variable `n`,
2) calculer le n-ième terme de la série harmonique,
3) afficher à l'écran le résultat du calcul.

Correction. Voir le fichier *Exo_20_harmonique.c*

4 Trucs et astuces !

4.1 Génération d'aléa

Exercice 21. Un peu de pratique.

Rédigez un code qui :

1) affiche un nombre entier aléatoire entre 0 et 2 inclus.
2) affiche un nombre entier aléatoire entre 1 et 3 inclus.
3) demande à l'utilisateurs deux valeurs `a` et `b` et affiche un nombre entier aléatoire entre `a` et `b` inclus.

Exercice 22. Un peu de pratique.

Sachant que l'opérateur `%` (modulo) permet de renvoyer de résultat du reste de la division euclidienne d'un entier par un autre, proposez une alternative à la génération de nombre aléatoire entre 0 et `n` puis entre `a` et `b`.

Exercice 23. Je pense encore à ....

L'objectif de cet exercice est de modifier le code de l'exercice 13 pour jouer seul. Modifiez donc le programme pour que :

1) l'utilisateur puisse fixer les limites du nombre mystère,
2) l'ordinateur génère aléatoirement le nombre entre les bornes définies,
3) l'utilisateur puisse modifier le nombre de tentatives autorisées.

Correction. Voir la correction dans le fichier *Exo_23_je_pense_encore_a.c*

Correction

4.2 scanf et les « retour charriot »

5 Exercices d'approfondissement

Exercice 24. Suite de Syracuse (Moodle).

La suite de Syracuse[1] est définie pour des entiers strictement positifs. Malgré son apparente simplicité, elle soulève de nombreux problèmes de prévision de comportement (comme par exemple la conjecture de Syracuse qui stipule que la suite de Syracuse de n'importe quel nombre entier naturel atteint 1). Le but de cet exercice est d'étudier quelques propriétés de cette afin d'infirmer ou confirmer cette hypothèse pour certaines valeurs.

La suite de Syracuse est définie par :

$$\begin{cases} u_0 & = c \\ u_{n+1} & = \begin{cases} u_n/2 \text{ si } u_n \text{ est pair,} \\ 3u_n + 1 \text{ sinon.} \end{cases} \end{cases}$$

Vous devez :

1) demander à l'utilisateur la valeur c permettant de définir u_0.
2) calculer puis afficher le n-ième terme de la suite en ayant demandé à l'utilisateur la valeur n.
3) afficher le temps de vol de la suite pour la valeur $u_0 = c$ (plus petit n tel que $u_n = 1$).
4) afficher l'altitude maximale de la suite pour la valeur $u_0 = c$ ($\max(u_k), 0 \leq k \leq n$).

Correction. Voir la correction dans le fichier *Exo_24_Syracuse.c*

Exercice 25. Stars Matrix like.

On se propose dans cet exercice de faire défiler sur la console des lignes d'étoiles aléatoirement remplies du bas vers le haut comme illustré ci-après :

```
**  **  *  ****  ***  *    * ****   * *** **  ** ***    * ****
  *  *   * ** ***    ** *** * *  ****   ******** * **  * **  ** *
**** *** *  * *    **    *  *  *   *     **  **** ***  *     **
 * * * ***     *** *****    ***** *** **   ****** *****  **    *
   *  *  ** * *  * *** ** * *  **  **    **  ** *  * *****   * *
  * ***   * *  * * *  * ** * **  ** *** ** * **** *  ** **    ***
** ** *  ***  *** *** ** * **  *** **      ***** ****** * *
  *   * *   *  * ****** *   *** *  ***** * *  *  *  * ****   **
***   ****  *     *    *** **** ** *  *   **   *   * ** *    **
 *   ** *  **  *** * ** * * **   *** *  *  *** *******      *
  *** **    * *  * *** ** * * * ** *  *   * ** ****** ** *     **
* *  *   *     *   *    **  ** * ****    ****   **    *   *  *
```

Pour cela, faire un programme qui :

1) fasse un boucle infinie,
2) dans cette boucle infinie pour chaque ligne de largeur `NbCol`, afficher aléatoirement pour chaque colonne soit le caractère `*`, soit le caractère espace.

1. `http://fr.wikipedia.org/wiki/Conjecture_de_Syracuse`

Correction

Petite variante : sur nos machines actuelles, l'affichage des lignes est très rapide. Si vous souhaitez ralentir l'affichage des lignes, il faut pour cela ralentir l'exécution du programme. Vous pouvez utiliser la fonction `usleep`. Voir le `man` pour plus d'explications.

Correction. Voir la correction dans le fichier *Exo_25_StarsMatrix.c*

Exercice 26. Le juste prix.

Vous connaissez sûrement le jeu de la vitrine dans le Juste Prix : un candidat doit trouver le montant exact de la vitrine sachant que ce dernier est entre 10000 et 50000 euros. Le candidat dispose de 45 secondes et à chaque proposition de prix, l'animateur répond `plus` ou `moins`. En utilisant l'exercice « je pense à », implémentez le jeu du juste prix en utilisant la bibliothèque `time.h` pour la gestion du temps et la fonction `time()`.

Vous pouvez également inverser les rôles, c'est à dire, l'utilisateur pense à un nombre et l'ordinateur tente de deviner lequel. Pour accélérer la réponse de l'utilisateur vous pouvez spécifier comme réponse uniquement '`+`' et '`-`'.

Correction. Voir la correction dans le fichier *Exo_26_juste_prix.c*

Exercice 27. Rendu de monnaie.

Un peu plus difficile que le simple rendu de monnaie, votre programme va devoir également calculer le nombre de chaque pièce et de chaque billet à rendre. Par exemple pour rendre 1,83 euros, vous devez afficher :

- 1 x 1 euros
- 1 x 50 cents
- 1 x 20 cents
- 1 x 10 cents
- 1 x 2 cents
- 1 x 1 cents

Pour simplifier vous pouvez commencer par utiliser uniquement les valeurs 100, 10, 1 euros et 10 et 1 centimes puis ajouter les autres valeurs par la suite.

Correction. Voir la correction dans le fichier *Exo_27_monnaie.c*