Quelques exercices de programmation en Perl

Alexandre Meslé

14 mars 2014

Table des matières

1 Exercices **2**
1.1 Introduction 2
1.1.1 Géométrie 2
1.1.2 Arithmétique 4
1.1.3 Passage de tableaux en paramètre 5
1.1.4 Décomposition en facteurs premiers 5

Chapitre 1

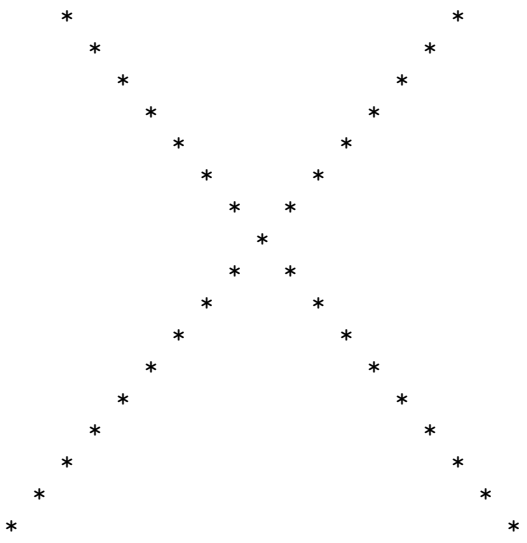
Exercices

1.1 Introduction

1.1.1 Géométrie

Ecrire un programme prenant en argument une valeur numérique positive n par la ligne de commande, et affichant un carré, une croix et un losange, tous de coté n. Par exemple, si $n = 10$, l'exécution donne

```
Saisissez la taille des figures
10
* * * * * * * * * *
*                 *
*                 *
*                 *
*                 *
*                 *
*                 *
*                 *
*                 *
* * * * * * * * * *
                  *
                *   *
              *       *
            *           *
          *               *
        *                   *
      *                       *
    *                           *
  *                               *
*                                   *
  *                               *
    *                           *
      *                       *
        *                   *
          *               *
            *           *
              *       *
                *   *
                  *
*                                   *
  *                               *
```



Vous définirez des sous-programmes de quelques lignes et au plus deux niveaux d'imbrication. Vous ferez attention à ne jamais écrire deux fois les mêmes instructions. Pour ce faire, complétez le code source suivant :

```
#!/usr/bin/perl

use strict;
use warnings;
use Carp ();
local $SIG{__WARN__} = \&Carp::cluck;

# affiche le caractère $c
sub afficheCaractere# char $c
{
}

# affiche $n fois le caractère $c
# ne revient pas à la ligne après.
sub ligneSansReturn# int $n, int $c
{
}

# affiche $n fois le caractère $c
# revient à la ligne après.
sub ligneAvecReturn# int $n, int $c
{
}

# affiche n espaces, revient à la ligne après.
sub nEspaces# int $n
{
}

# affiche $c à la colonne $n, affiche des espaces avant.
# ne revient pas à la ligne après
# les indices commencent à 1.
sub unCaractereSansReturn# int $n, int $c
{
}

# affiche $c à la colonne $n, affiche des espaces avant.
# revient à la ligne après
sub unCaractereAvecReturn# int $n, int $c
{
}

# affiche $c aux colonnes $i et $j
# revient à la ligne après.
sub deuxCaracteres# int $i, int $j, char $c
{
}

# affiche un carré de coté $n.
sub carre# int $n
```

```
{
}

# affiche un chapeau dont la pointe (non affichée)
# est sur la colonne $centre en utilisant des $c.
sub chapeau# int $centre, char $c.
{
}


# affiche un chapeau retourné dont la pointe (non affichée)
# est sur la colonne $centre en utilisant des $c.
sub chapeauInverse# int $centre, char $c.
{
}

# affiche un losange de coté $n.
sub losange# int $n
{
}

# affiche un losange de coté $n.
sub croix# int $n
{
}

carre $ARGV[0];
losange $ARGV[0];
croix $ARGV[0];
```

1.1.2 Arithmétique

Exercice 1 - chiffres et nombres

1. Ecrire la fonction `unites(n)` retournant le chiffre des unités du nombre n.
2. Ecrire la fonction `dizaines(n)` retournant le chiffre des dizaines du nombre n.
3. Ecrire la fonction `extrait(n, p)` retourant le p-ème chiffre de représentation décimale de n en partant des unités.
4. Ecrire la fonction `nbChiffres(n)` retournant le nombre de chiffres que comporte la représentation décimale de n.
5. Ecrire la fonction `sommeChiffres(n)` retournant la somme des chiffres de n.

Exercice 2 - Nombres amis

Soient a et b deux entiers strictement positifs. a est un diviseur strict de b si a divise b et $a \neq b$. Par exemple, 3 est un diviseur strict de 6. Mais 6 n'est pas un diviseur strict 6. a et b sont des nombres amis si la somme des diviseurs stricts de a est b et si la somme des diviseurs de b est a. Le plus petit couple de nombres amis connu est 220 et 284.

1. Ecrire une fonction `sommeDiviseursStricts(n)`, elle doit renvoyer la somme des diviseurs stricts de n.
2. Ecrire une fonction `sontAmis(a, b)`, elle doit renvoyer 1 si a et b sont amis, 0 sinon.

Exercice 3 - Nombres parfaits

Un nombre parfait est un nombre égal à la somme de ses diviseurs stricts. Par exemple, 6 a pour diviseurs stricts 1, 2 et 3, comme $1 + 2 + 3 = 6$, alors 6 est parfait.

1. Est-ce que 18 est parfait ?
2. Est-ce que 28 est parfait ?
3. Que dire d'un nombre ami avec lui-même ?
4. Ecrire la fonction `estParfait(n)`, elle doit retourner 1 si n est un nombre parfait, 0 sinon.

Exercice 4 - Nombres de Kaprekar

Un nombre n est un nombre de Kaprekar en base 10, si la représentation décimale de n^2 peut être séparée en une partie gauche u et une partie droite v tel que $u+v=n$. $45^2 = 2025$, comme $20+25=45$, 45 est aussi un nombre de Kaprekar. $4879^2 = 23804641$, comme $238 + 04641 = 4879$ (le 0 de 046641 est inutile, je l'ai juste placé pour éviter toute confusion), alors 4879 est encore un nombre de Kaprekar.

1. Est-ce que 9 est un nombre de Kaprekar ?
2. Ecrire la fonction `sommeParties(n, p)` qui découpe n est deux nombres dont le deuxième comporte p chiffres, et qui retourne leur somme. Par exemple,

$$sommeParties(12540, 2) = 125 + 40 = 165$$

3. Ecrire la fonction `estKaprekar(n)`

1.1.3 Passage de tableaux en paramètre

Exercice 5 - Somme

Ecrire une fonction `somme(t)` retournant la somme des éléments du tableau t (vous remarquerez que t peut aussi être une liste de scalaires).

Exercice 6 - Minimum

Ecrire une fonction `min(t)` retournant la valeur du plus petit élément de t.

Exercice 7 - Recherche

Ecrire une fonction `existe(k, t)` retournant `vrai` si et seulement si k est un des éléments de T.

Exercice 8 - Somme des éléments pairs

Ecrire une fonction `sommePairs(t)`, retournant la somme des éléments pairs de t. N'oubliez pas que $a\%b$ est le reste de la division entière de a par b.

Exercice 9 - Vérification

Ecrire une fonction `estTrie(t)`, $estTrie(t)$ retournant vrai si et seulement si les éléments de t sont triés dans l'ordre croissant.

Exercice 10 - Permutation circulaire

Ecrire une fonction `permutation(t)` effectuant une permutation circulaire vers la droite des éléments de t.

Exercice 11 - Miroir

Ecrire une fonction `miroir(t` inversant l'ordre des éléments de t.

1.1.4 Décomposition en facteurs premiers

On rappelle qu'un nombre est premier s'il n'est divisible que par 1 et par lui-même. Par convention, 1 n'est pas premier.

Exercice 12

Écrivez une fonction `estPremier(x, premiers)` retournant vrai si et seulement si x est premier. Vous vérifierez la primarité de x en examinant les restes des divisions de x par les éléments de $premiers$.

Exercice 13

Modifiez la fonction précédente en tenant compte du fait que si aucun diviseur premier de x inférieur à $\sqrt{x}$ n'a été trouvé, alors x est premier

Exercice 14

Écrivez une fonction `trouvePremiers(n)` retournant un tableau contenant les n premiers nombres premiers.

Exercice 15

Écrivez une fonction `decompose(x, premiers)` prenant en paramètre un entier et une référence vers un tableau. *decompose* retourne une référence vers un tableau contenant la décomposition en facteurs premiers du nombre x, sachant que *premiers* contient les premiers nombres premiers.

Par exemple, si $x = 108108$, alors on décompose n en produit de facteurs premiers de la sorte

$$108108 = 2 * 2 * 3 * 3 * 7 * 11 * 13 = 2^2 * 3^3 * 5^0 * 7^1 * 11^1 * 13^1 * 17^0 * 19^0 * \ldots * Z^0$$

(où Z est le n-ième nombre premier). On représente donc x de façon unique par le tableau à n éléments suivant :

$$\{2, 3, 0, 1, 1, 1, 0, 0, 0, \ldots, 0\}$$

Exercice 16

Écrivez une fonction `recompose(decomposition, premiers)` effectuant l'opération réciproque de celle décrite ci-dessus.

Exercice 17

Écrivez une fonction `pgcdTab(a, b)` prenant en paramètre les références des décompositions en facteurs premiers a et b de deux nombres, et retournant une référence de la décomposition en facteurs premiers du plus grand commun diviseur de a et b.

Exercice 18

Écrivez une fonction `pgcd(a, b)` prenant en paramètres deux nombres a et b, et combinant les fonctions précédentes pour retourner le *pgcd* de a et b.