

Bases de la Programmation

TP 2 – boucles **while** et **do-while**

V. Padovani, PPS - IRIF

Exercice 1 :

Écrire un programme affichant indéfiniment un message à l'écran ¹.

Exercice 2 : Suite de Syracuse

Pour tout entier naturel m , la *suite de Syracuse* engendrée par m est la suite $S_m = (m_i)$ définie de la manière suivante :

$$\begin{aligned} m_0 &= m \\ m_{i+1} &= m_i/2 && \text{si } m_i \text{ est pair} \\ m_{i+1} &= 3m_i + 1 && \text{sinon.} \end{aligned}$$

Par exemple :

$$\begin{aligned} S_6 &= (6, 3, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, \dots) \\ S_{11} &= (11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, \dots) \end{aligned}$$

La *conjecture de Collatz* est la suivante : pour tout entier m , la suite (S_m) atteint invariablement 1. Le *temps de vol* de m est le plus petit entier n tel que S_m vaille 1. La conjecture de Collatz est donc : le temps de vol de tout entier est défini.

1. Écrire un programme lisant un entier m au clavier, puis affichant son temps de vol. Testez-le sur différentes valeurs, par exemple 27.
2. Ecrire un programme vérifiant la conjecture de Collatz pour les entiers compris entre 1 et 100.

Exercice 3 :

1. Écrire un programme lisant des entiers au clavier jusqu'à ce que l'utilisateur tape un 0, puis affichant la somme de toutes les valeurs non nulles lues.
2. Modifier le programme pour qu'il affiche la somme et le produit des valeurs non nulles lues.
3. Modifier le programme pour qu'il affiche aussi le nombre de valeurs non nulles lues, ainsi que la moyenne (en **double**) de ces valeurs.

Remarque. Cet exercice se résout plus naturellement avec un **do-while** qu'un **while** – on souhaite effectuer au moins une lecture, quitte à ce que le nombre lu soit immédiatement nul.

1. L'interruption d'un programme se fait par ctrl-c dans le terminal.

Exercice 4 : Le nombre juste, utilisateur contre machine

Écrire un programme choisissant au hasard un nombre s compris entre 1 et 100 inclus (*c.f.* le TP 1), puis demandant à l'utilisateur de deviner ce nombre. À chaque fois que l'utilisateur entrera un nombre n ,

- si $n < s$, le programme affichera « Trop petit ! » et redemandera un nouveau nombre ;
- si $n > s$, le programme affichera « Trop grand ! » et redemandera un nouveau nombre ;
- si $n = s$, le programme affichera « Gagné ! » suivi du nombre d'essais de l'utilisateur, puis terminera.

Exercice 5 : Le nombre juste, machine contre utilisateur

Écrire un programme devinant un nombre entre 1 et 100 choisi mentalement par l'utilisateur. Le programme utilisera 3 variables **p** (petit), **m** (moyen) et **g** (grand). À chaque étape, le programme demandera à l'utilisateur si la valeur courante de **m** est l'entier choisi ; l'utilisateur devra répondre -1 (trop petit), 0 (juste) ou 1 (trop grand).

Initialement, **p** vaut 1 et **g** vaut 101. À chaque étape,

- **m** est (re)défini comme la partie entière de $(\mathbf{p} + \mathbf{g}) / 2$.
- si **m** est le nombre cherché, le programme termine ;
- sinon, et si **m** est trop petit, **p** est redéfini comme $\mathbf{m} + 1$
- sinon, **g** est redéfini comme **m**.

Votre programme devra terminer en signalant une erreur si **p** devient égal à $\mathbf{g}^2$.

Exercice 6 : Casse-tête n° 1

À chaque (ré)-évaluation, l'expression `time(NULL)` vaut l'heure courante mesurée en secondes depuis le premier janvier 1970 (le programme doit mentionner dans ce cas la directive `#include <time.h>`). Reprendre le programme de l'exercice 1. Modifier ce programme pour qu'il affiche votre message pendant 5 secondes, puis termine.

Exercice 7 : Casse-tête n° 2

Écrire un programme lisant un entier au clavier, puis affichant son nombre de chiffres. Votre programme ne devra pas utiliser d'autre type que `int`.

2. Dans ce cas, l'une au moins de réponses de l'utilisateur était incorrecte.