Programmation C
TP n° 1 : Introduction

> *Despite some aspects mysterious to the beginner and occasionally even to the adept, C remains a simple and small language, translatable with simple and small compilers. Its types and operations are well-grounded in those provided by real machines, and for people used to how computers work, learning the idioms for generating time- and space-efficient programs is not difficult. At the same time the language is sufficiently abstracted from machine details that program portability can be achieved.*
>
> — *Dennis M. Ritchie (1993).*

Compilation

Pour cette séance, la compilation de votre code se fera avec le `Makefile` minimal vu en cours, en le complétant à chaque nouvel exercice traité. Par exemple, après les deux premiers, votre fichier `Makefile` devrait être de la forme suivante :

```
CC=gcc
CFLAGS=-Wall -g

ALL=exo1 exo2
all: $(ALL)

exo1: exo1.c
exo2: exo2.c

clean:
	rm -rf $(ALL)
```

Noter que toute ligne indentée de ce fichier (ici la ligne avec `rm`) commence par un caractère de tabulation et non par une suite d'espaces. N'oubliez pas de terminer la dernière ligne par un retour à la ligne.
Le choix de l'éditeur est libre, mais nous vous suggérons d'utiliser `emacs` dont la tabulation automatique facilitera la détection d'erreurs de syntaxe de base.

Affichages

La fonction prédéfinie `printf` permet l'affichage de chaînes de caractères, ainsi que l'insertion de une ou plusieurs valeurs d'expressions dans ces chaînes. L'usage de cette fonction impose de commencer le programme avec comme première ligne :

```
#include<stdio.h>
```

Les expressions insérées peuvent être réduites à une variable, ou combiner des variables ou des constantes à l'aide d'opérateurs. Les points d'insertions s'écrivent `%d` pour des expressions à valeurs entières. Chaque `\n` rencontré dans la chaîne entraîne un retour à la ligne.

Par exemple, si `x` et `y` sont deux variables de `main` de type `int` et de valeurs `42` et `2`, on peut écrire dans `main` :

```
1    printf("%d\n", x + 1);                  // affiche "43"
2    printf("%d + %d = %d\n", x, y, x + y); // affiche "42 + 2 = 44"
```

Boucles simples et imbriquées

Exercice 1 : Suite de Syracuse

Étant donné un entier naturel m, la *suite de Syracuse* (S) engendrée par m est définie de la manière suivante :

- $S_0 = m$,
- si S_i est pair : $S_{i+1} = S_i/2$,
  sinon : $S_{i+1} = (3 \times S_i) + 1$.

Le *temps de vol* de m est le plus petit entier i tel que $S_i = 1$. La *conjecture de Collatz* est que pour tout entier m, la suite de Syracuse engendrée par m atteint toujours 1, *i.e.* le temps de vol de tout entier naturel est fini.

1. Ecrire un programme calculant et affichant les premiers termes de la suite de Syracuse d'une constante `N` (*e.g.* 27) déclarée en début de programme par un `#define`, jusqu'à ce que cette suite atteigne la valeur 1 (noter que la simple terminaison du programme est la preuve de la conjecture pour cette constante). Les valeurs de cette suite seront affichées sur des lignes distinctes :

   ```
   27
   82
   41
   124
   62 ...
   ```

2. Compléter le programme de manière à calculer simultanément le temps de vol de `N`. Ce temps de vol sera stocké dans une variable dont on affichera la valeur en fin d'éxécution, après celle de `N` :

   ```
   ...
   27 : 111
   ```

3. Modifier le programme afin de vérifier la conjecture de Collatz pour tous les entiers naturels de `1` à `N`. Le programme affichera simplement, sur des lignes distinctes, chaque entier suivi de son temps de vol.

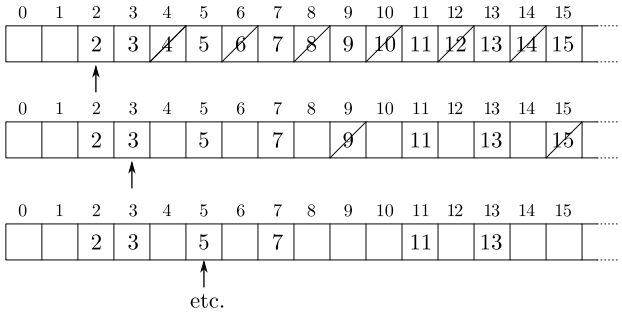
   ```
   1 : 0
   2 : 1
   3 : 7
   4 : 2
   5 : 5
   6 : 8
   7 : 16
   8 : 3 ...
   ```

Exercice 2 : Crible d'Ératosthène

Le *crible d'Ératosthène* est une méthode permettant de calculer tous les nombres premiers inférieurs à un entier *SUP* donné. Son principe est le suivant :

- On écrit la liste de tous les entiers supérieurs ou égaux à 2 et inférieurs à *SUP*.
- On effectue un parcours de cette liste. À chaque entier i rencontré, on supprime de la liste toutes les entiers strictement plus grands que i et multiples de i encore présents.

En fin de traitement, les nombres encore présents dans la liste sont tous les nombres premiers inférieurs à *SUP*. Voici par exemple l'état de la liste après la rencontre de 2, puis 3 – à la dernière étape, la rencontre de 5 sera suivie de l'effacement de tous les multiples de 5 encore présents (*e.g.* 25), etc.



Implémentez cette méthode en C. Votre programme devra isoler et tous les nombres premiers strictement inférieurs à une constante `SUP` de valeur supérieure ou égale à 2. En fin de traitement, ces nombres seront inscrits dans un tableau de taille `SUP` : chaque position `i` du tableau contiendra `i` si ce nombre est premier, et `0` sinon (la notion d'"effacement d'un nombre i de la liste" de la méthode précédente sera donc traduite en l'inscription d'un `0` à la position `i`). Les valeurs non nulles du tableau seront simultanément affichées pendant sa construction :

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 ...
```

Remarque. Un entier j est multiple de i si et seulement si j modulo i est égal à 0. L'opérateur de modulo s'écrit `%` en C.