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).*

Environnement de développement vs. compilation

Un programme écrit en langage `C` doit être compilé afin de produire un programme exécutable par votre machine. Au cours des TP, vous aurez le choix :

- d'utiliser un environnement de développement vous permettant d'éditer, compiler et exécuter vos programmes,
- ou bien de compiler et exécuter votre programme depuis votre shell.

Environnement de développement (IDE)

Nous vous proposons d'utiliser l'IDE *Geany* (`https://www.geany.org/`) qui est compatible avec les systèmes d'exploitation Linux, MacOs et Windows.
Une fois *Geany* installé sur votre machine, vous procédez de la façon suivante :

1. dans le menu `Fichier`, choisir `Nouveau` pour éditer un nouveau fichier dans lequel vous écrirez votre programme. Vous pouvez ensuite sauvegarder ce fichier en sélectionnant dans le menu `Fichier`, `Enregistrer sous` ;
2. une fois le programme écrit, vous pouvez le compiler et construire le fichier exécutable en cliquant sur le bouton `Construit le fichier courant` ( ) ;
3. enfin pour exécuter ce fichier exécutable, cliquer sur `Exécuter ou voir le fichier courant` ( ).

Compilation et exécution en ligne de commande

Vous pouvez si vous ne souhaitez pas utiliser d'IDE, éditer votre programme avec votre éditeur préféré (`emacs`[1], `vim`, ...), le compiler dans votre shell avec la ligne de commande suivante :
`gcc -Wall` *monprog.c* `-o` *monprog*
où *monprog.c* est le nom de votre programme `C` et *monprog* est le nom de l'exécutable créé à la compilation. L'option `-Wall` permet l'affichage des messages « warnings ». Ces messages signalent des problèmes qui n'empêchent pas la compilation mais qu'il ne faut en général pas ignorer. Vous pouvez ensuite lancer cet exécutable depuis votre shell en invoquant la commande suivante (bien sûr depuis le répertoire où se trouve l'exécutable) :
`./monprog`

1. `emacs` fait des tabulations automatiques, ce qui facilite la détection d'erreurs de syntaxe de base.

Premiers pas en C

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 d'inclure le fichier d'en-tête `stdio.h` en commençant 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 la fonction principale `main` de type `int` et de valeurs `42` et `2`, on peut écrire dans `main` :

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

```

Le manuel

Pour les fonctions `C`, vous disposez d'un manuel en ligne de commande, accessible avec la commande `man` :
`man` *num fct*
affiche le manuel numéro *num* de la fonction *fct*. Les fonctions `C` sont définies dans les manuels 2 (fonctions effectuant des appels systèmes) et 3 (fonctions de la bibliothèque `C`). Par exemple `man 2 time` affiche le manuel de la fonction `C time`, et `man 3 rand` affiche le manuel de la fonction `C rand`. Vous trouverez en plus du prototype et de la description de la fonction, les fichiers d'en-tête à inclure pour pouvoir appeler la fonction dans un programme `C`.

Exercice 1 : premier programme C

1. Écrire un programme `bonjour.c` qui affiche la phrase suivante avec retour à la ligne « Bonjour, ceci n'est pas un programme java ! »
2. Modifier votre programme pour qu'il affiche également le contenu d'une variable entière `z` à laquelle vous aurez donné une valeur au préalable. Le programme afichera donc en plus « La variable z a pour valeur *val* » où *val* est la valeur de `z`.

Exercice 2 : aléa

Pour obtenir un nombre (pseudo) aléatoire en `C`, vous pouvez utiliser la fonction `int rand()` qui retourne un entier aléatoire compris entre 0 et `RAND_MAX` (qui vaut au moins 32767).
Il faut commencer par initialiser la suite de nombres aléatoires en appelant la fonction `void srand(unsigned seed)` avec par exemple comme graine de départ l'heure courante :

```
srand(time(NULL));
```

Ensuite, à chaque appel de la fonction `rand`, celle-ci retourne un nouveau nombre aléatoire.

1. Écrire un programme `alea.c` qui tire un nombre aléatoire et affiche si celui-ci est pair ou impair.
   Remarque : l'opérateur de modulo s'écrit `%` en `C`.
2. Ajouter à votre programme une fonction `stat` qui prend deux paramètres entiers `m` et `n`, tire des nombres aléatoires dans l'intervalle `[0, k]`, où k est la partie entière de `n+m/2`, tant que l'un deux n'est pas strictement supérieur à `n` et affiche ensuite le nombre et le pourcentage d'entiers aléatoires tirés dans l'intervalle `[0, m[` et dans l'intervalle `[m, n]`.
   Si `m > n`, la fonction affichera un message d'erreur.
3. Tester la fonction `stat`.

Exercice 3 : tableau

1. Écrire un programme `tableau.c` qui crée un tableau de `M` entiers positifs aléatoires de l'intervalle `[0, N]` où `M` et `N` sont des constantes entières positives.
2. Ajouter au programme une fonction `aff_tab` qui affiche un tableau passé en paramètre.
3. Ajouter au programme une fonction `max` qui retourne l'indice de l'élément le plus grand du tableau passé en paramètre. Si plusieurs éléments sont maximaux, alors l'indice retourné est le plus petit, et si le tableau vide alors la fonction retournera -1.

Exercice 4 : 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) 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.

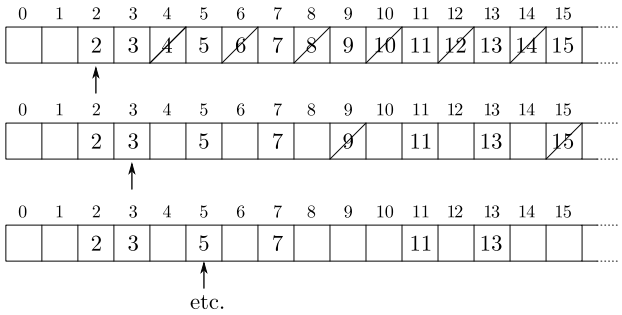
```
1 : 0
2 : 1
3 : 7
4 : 2
5 : 5
6 : 8
7 : 16
8 : 3 ...
```

Exercice 5 : 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 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.