Programmation C
TP n° 1 : Introduction

Les exercices indiqués avec une $*$ sont à rendre.

Compilation de vos programmes

Un programme écrit en langage `C` doit être compilé pour produire un exécutable. Pour les TP de ce cours, nous vous laissons le choix entre :

- écrire vos programmes dans un éditeur, et les compiler et les exécuter depuis le shell,
- ou utiliser un environnement de développement vous permettant d'éditer, compiler et exécuter vos programmes avec une unique interface.

Compilation et exécution en ligne de commande

L'utilisation d'un éditeur tel qu'`emacs` pour l'écriture de vos programmes offre un avantage que ne proposent pas tous les IDE : celui de la tabulation automatique du code au fur et à mesure de son écriture, qui facilite la détection d'erreurs de syntaxe de base. (*e.g.* oublis de points-virgules, de parenthèses, d'accolades, de guillemets fermants, etc.).
La compilation en ligne de commande permet d'autre part de forcer le compilateur à signaler tous les éléments du code qui, sans rendre le programme incompilable, lui semblent suspects (*e.g.* erreurs de typage probables, parties du code inutiles ou manquantes, etc).
Le fichier de votre programme doit avoir l'extension `.c`. Une fois sauvegardé sous `emacs`, pour compiler et exécuter ce programme, placez-vous dans le terminal dans le répertoire de sauvegarde et entrez la commande suivante :

`gcc -Wall -o` *monprog monprog.c*

où *monprog.c* est le nom du fichier et *monprog* le nom de l'exécutable à créer (l'option `-Wall` force l'affichage des « warnings », les messages d'alerte indiquant des erreurs probables). Pour lancer l'exécutable, invoquez dans le même répertoire la commande suivante (l'ajout de `./` avant le nom de l'exécutable, sans aucun espace, est indispensable) :

`./`*monprog*

Environnement de développement (IDE)

Si vous préférez programmer dans un IDE, vous pouvez par exemple vous servir de *Geany* (`https://www.geany.org/`), compatible avec les systèmes d'exploitation Linux, MacOs et Windows. Une fois installé et lancé sur votre machine, Geany s'utilise de la manière suivante :

1. Dans le menu `Fichier`, choisir `Nouveau` pour éditer un nouveau fichier dans lequel vous écrirez votre programme. Vous pourrez sauvegarder ce fichier en sélectionnant dans le menu `Fichier` : `Enregistrer sous`.
2. Pour compiler le fichier, cliquez sur le bouton : `Construit le fichier courant` ( ).
3. Pour lancer l'exécutable, cliquer sur : `Exécuter ou voir le fichier courant` ( ).

Affichage et lecture de valeurs

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. Pour vous servir de cette fonction (et de la fonction `scanf` ci-dessous), ajoutez en première ligne de votre programme :

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

Les expressions insérées peuvent être des variables, des constantes ou des expressions combinées à l'aide d'opérateurs. Les points d'insertions s'écrivent `%d` pour des expressions à valeurs entières, et `%lf` pour des valeurs de type `double`. Chaque `\n` rencontré dans la chaîne entraîne un retour à la ligne. On peut par exemple écrire :

```
  #include <stdio.h>
  #include <stdlib.h>
  int main() {
    unsigned a=3;
    int n = 42;
    int m = 2;
    double x = 3.14;
                                          // (dans le terminal :)
    printf("%u\n",a);                     //3
    printf("%d\n", n + 1);                // 43
    printf("%d + %d = %d\n", n, m, n + m);   // 42 + 2 = 44
    printf("%d / %lf = %lf\n", n, x, n / x); // 42 / 3.140000 = 13.375796

   return EXIT_SUCCESS;
  }
```

Le code ci-dessous montre la manière dont on peut lire une valeur de variable entière au clavier, à l'aide de la fonction prédéfinie `scanf`. L'appel de `scanf` suspendra temporairement l'exécution du programme en redonnant la main au terminal. L'utilisateur pourra alors rentrer un nombre, suivi d'un retour-chariot (touche "Entrée") : le programme reprendra alors la main, et stockera le nombre lu dans la variable `v`.

```
#include <stdio.h>
#include <stdlib.h>
int main () {
  unsigned v;

  scanf ("%u", &v);
  printf ("v : %u\n",v);

  return EXIT_SUCCESS;
}
```

Notez le symbole `&` précédant le nom de la variable. Le sens cet opérateur vous sera expliqué plus tard. La variable `v` ne prendra une valeur que si l'utilisateur rentre effectivement une suite de caractères formant un nombre entier signé en décimal et restera de valeur indéfinie sinon.

Exercice 1 : Premier programme

Pour être simplement sûr que vous êtes prêts : compilez sans erreur puis exécutez un premier

programme affichant un message d'accueil dans le terminal. Complétez le code de manière à lire les valeurs de deux variables emtières au clavier, et affichez leurs valeurs et leur somme.

Exercice 2 : Boucles simples et imbriquées (∗)

1. Écrire un programme lisant un entier `n` positif au clavier et affichant la somme des cubes des `n` premiers entiers naturels positifs. Par exemple, si l'utilisateur entre `5`, votre programme devra afficher `225` :

$$\sum_{k=1}^{5} k^3 = 1^3 + 2^3 + 3^3 + 4^3 + 5^3 = 225$$

2. Ecrire une fonction `unsigned fact (unsigned n)` calculant et renvoyant la *factorielle* de `n` (supposé positif). On a par définition $0! = 1$, et pour tout $n \geqslant 1$:

$$n! = 1 \times 2 \times \ldots \times n$$

   Compléter le code par un `main` lisant un entier positif au clavier, et affichant la factorielle de cet entier.

3. Modifier la fonction `fact` précédente afin qu'elle affiche, en une seule boucle, les factorielles de chaque entier compris entre `1` et `n` :

```
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
...
```

4. Écrire un programme qui affiche la moyenne d'une suite d'entiers strictement positifs entrés au clavier. On arrêtera la saisie quand l'utilisateur entrera le nombre 0. Par exemple :

```
Entrez un entier : 2
Entrez un entier : 5
Entrez un entier : 0
La moyenne vaut : 3.500000
```

Exercice 3 : Suite de Syracuse (∗)

Étant donné un entier $n \geqslant 1$, la *suite de Syracuse* engendrée par n est la suite des valeurs prises par n dans le traitement suivant :

1. Si n n'est pas égal à 1, alors :
   (a) si n est pair, on le divise par 2,
   (b) sinon, on multiplie n par 3 et on lui ajoute 1,
   on revient à l'étape 1.

Le *temps de vol* de n est le nombre de fois où l'étape 1 est effectuée dans ce traitement. Le temps de vol de 1 vaut 0. Le temps de vol de 3 vaut 7, la suite de Syracuse de 3 étant :

$3, 10, 5, 16, 8, 4, 2, 1$.

La *conjecture de Collatz* est que pour tout entier n, la suite de Syracuse engendrée par n atteint toujours 1, *i.e.* le temps de vol de tout entier naturel est fini.

1. Ecrire un programme calculant et affichant la suite de Syracuse d'une constante `N` définie par un `#define` en début de programme (*e.g.* `27`) (noter que la simple terminaison du programme est la preuve de la conjecture pour cette constante). Les valeurs de cette suite seront affichées en les séparant par des espaces :

   ```
   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, sur une nouvelle ligne, et 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 4 : S'entraîner encore

Pour toutes les fonctions que l'on vous demande dans cet exercice, n'oubliez pas de les tester avec votre programme principal.

1. Écrire une fonction `unsigned retourne(unsigned n)` qui renvoie le nombre de chiffres de `n`. Par exemple, `retourne(3567)` renverra `4`.
2. Écrire une fonction `unsigned sommeChiffre(unsigned n)` qui renvoie la somme des chiffres de `n`.
3. On dit qu'un entier n est un *nombre narcissique* si la somme des cubes de ces chiffres est égal à n. Par exemple : $153 = 1^3 + 5^3 + 3^3$ est un nombre narcissique.
   (a) Écrire une fonction `bool estNarcissique(unsigned n)` qui renvoie `true` si `n` est narcissique est `false` sinon (n'oubliez pas d'inclure `<stdbool.h>` pour pouvoir utiliser le type `bool`).
   (b) Écrire une fonction `void afficheNarcissique(unsigned n)` qui affiche tous les nombres narcissiques inférieurs ou égaux à `n`.
4. Écrire une fonction `unsigned retourne(unsigned n)` qui renvoie un nombre correspondant au nombre `n` lu à l'envers. Par exemple, `retourne(123)` renverra `321`.