# TP 0 – rappels

Programmation et systèmes, L2 informatique 2017-2018

jean.connier@uca.fr

## Préambule

Les questions que vous devez vous poser à peu près à chaque exercice :

- Qu'est-ce que cette fonction et censée faire ?
- Qu'est-ce que la fonction/programme/etc. prend en **entrée** ?
- Qu'est-ce que la fonction/programme/etc. donne en **sortie** ?

**Ensuite**, vous pourrez vous demander :

- Comment le faire ?

**Ensuite**, vous pourrez vous demander :

- Comment *mieux* le faire ?

**Ensuite**, quand tout ça est assez clair, vous pouvez **commencer à programmer**. Si vous commencez à programmer tout de suite, vous allez probablement vous embrouiller et perdre du temps (et il y a trente-neuf exercices quand même).

## Exercices

**Exercice 1** Faire un programme qui affiche "hello world".

**Exercice 2** Faire une fonction qui s'appelle **hello** et qui affiche "hello world". Qu'est-ce qu'elle prend en paramètre ? Qu'est-ce qu'elle retourne ? Choisissez avec discernement.

**Exercice 3** Faire un programme qui utilise la fonction **hello** pour afficher **n** fois "hello world". **n** est une variable locale, du type *le plus approprié*.

**Exercice 4** Faire une fonction qui prend en argument une chaîne de caractères et qui renvoie sa longueur (nombre de caractères). N'utilisez pas la libc.

**Exercice 5** Faire une fonction (appellée "**multiplier**") qui prend trois arguments : deux nombres à virgule à multiplier et le résultat. La fonction ne retourne rien avec *return*. Evidemment la fonction doit quand même donner un résultat; pour ça on rajoute un paramètre (pour ceux qui suivent pas au fond, on passe une adresse où écrire le résultat).

**Exercice 6 (bonus)** Faire une fonction ("**string_to_int**") qui prend en argument une chaîne de caractère et qui renvoie :

- Si la chaîne contient un entier (par exemple "21"), l'entier.
- Si la chaîne contient autre chose que des chiffres, -1.

**Exercice 7 (bonus)** Faire une fonction string_2_int qui fait la même chose que la précédente sauf que le *return* est seulement utilisé pour signaler le bon fonctionnement ou une erreur.

- 0 -> bon fonctionnement (la chaîne contient uniquement des chiffres);
- 1 -> erreur (la chaîne contient des non-chiffresa.)

**Exercice 8 (bonus)** Testez la fonction string_2_int sur *plusieurs* chaînes de caractères. Au minimum :

- "123"
- "abc"
- "12356489798756431321312654897"
- "12a"
- "a12"
- "000"
- "001"

A partir de maintenant, testez toutes vos fonctions ! Sinon, comment savoir si elles marchent ?

**Exercice 7** Maintenant lisez man 3 atoi.

**Exercice 8** (Re)faites la fonction de l'exercice 7 en utilisant atoi.

**Exercice 9** Ecrivez le programme **liste_arguments** qui liste les arguments qu'on lui donne (sur la ligne de commande). Indice :

- *argc* veut dire *arguments count*;
- *argv* veut dire *arguments value*.

**Exercice 10** Ecrivez la fonction **dessiner_sapin** qui... dessine un sapin. Elle prend en paramètre un *int* qui définit la taille (hauteur) du sapin à dessiner. Ceci est un sapin de taille 7 (les points représentent des espaces):

```
....*....
...***...
..*****..
.*******.
*********
...***...
...***...
```

**Exercice 11** Faites une fonction **hexamaj** prend en argument une chaîne de caractères et qui remplace les caractères 'a', 'b', 'c', 'd', 'e', 'f' par leurs versions capitales ('A', etc.) et le caractère 'X' par 'x'.

**Exercice 12** Faites un programme **sapin** qui affiche dans le terminal un sapin de la taille passée en paramètre.

**Exercice 13** Améliorez le programme **sapin** pour qu'il affiche une petite explication quand l'utilisateur ne donne pas le bon nombre d'arguments (c'est-à-dire un et un seul argument).

**Exercice 14** Ajoutez une fonctionnalité au programme sapin ! Maintenant l'utilisateur donne autant d'arguments qu'il souhaite ! Pour chaque argument, le programme dessinera un sapin de la taille contenue dans l'argument. Par exemple, si je tape

```
sapin 4 2 7
```

le programme affichera un sapin de taille 4, puis un sapin de taille 2, puis un sapin de taille 7.

**Exercice 15** Prenez en compte les erreurs dans le programme ("sapin 1 2 jksdfh").

**Exercice 16** Faites une fonction **inverse_chaine** qui inverse l'ordre des caractères d'une chaîne passée en paramètre. "abcdef" -> "fedcba".

**Exercice 17** Créez un type **matrice33** qui représente une matrice carrée 3x3.

**Exercice 17** Ecrivez la fonction **mul_matrice33** qui multiplie deux *matrice33* et met la résultat dans une *matrice33* passée par pointeur.

**Exercice 18** Créez un type **matrice_carree**. Une **matrice_carree** peut être de taille arbitraire. Il y a (au moins) deux manières de représenter une matrice avec un pointeur. Choisissez la version avec pointeur simple.

**Exercice 19** Ecrivez la fonction **creer_matrice_nulle** qui crée une *matrice_carree* de la taille spécifiée, remplie de 0. **creer_matrice_nulle** renvoit un pointeur vers la *matrice_carree* créée.

**Exercice 20** Ecrivez une fonction **supprimer_matrice_carree** qui prend un pointeur sur une *matrice_carree* et qui fait ce qu'il y a à faire (vous **savez** ce qu'il y a à faire).

**Exercice 21** Créez un type **liste_chainee_int**. Une variable de type **liste_chainee_int** contient en fait un élément d'une liste (simplement) chaînée de int (elle ne représente pas *toute* la liste chaînée). Un élément de liste contient deux choses :

- un int (évident, non ?);
- un pointeur vers le prochain élément de la liste (sinon la liste ne serait pas chaînée – et n'aurait probablement qu'un élément, ce serait donc une liste particulièrement inintéressante, et nous n'aimons pas les listes inintéressantes).

**Exercice 22** Ecrivez une fonction **ajouter_element** qui ajoute un élément à la fin de la liste.

**Exercice 23** Ecrivez une fonction (bien nommée) qui fait ce qu'il y a à faire sur la liste avant de terminer le programme (vous **SAVEZ** ce qu'il y a à faire).

**Exercice 24** Ecrivez une fonction **tableau_vers_liste** qui transforme un tableau i (de *int*) en *liste_chainee_int*.

**Exercice 25** Ecrivez une fonction **chercher** qui retourne l'adresse du premier élément d'une *liste_chainee_int* dont la valeur est celle recherchée. Pensez au cas où l'élément n'est pas dans la liste.

**Exercice 26** Ecrivez une fonction qui met les valeurs d'un tableau au carré.

**Exercice 27** Ecrivez une fonction qui affiche la taille de tous les types simple.

**Exercice 28** Créez un type etudiant. Un étudiant est composé d'un identifiant (entier positif) et d'une note (on peut avoir 14.67 (vous pouvez ?).

**Exercice 29** Ecrivez la fonction **afficher_etudiant** qui affiche un étudiant.

**Exercice 30** Ecrivez la fonction **creer_etudiant** qui crée un étudiant avec des caractéristiques aléatoires (lisez **man 3 rand**).

**Exercice 31** Ecrivez un programme qui contient au moins une variable de chacune de ces catégories :

- Variable globale;
- Variable automatique (locale) niveau fonction;
- Variable automatique (locale) de bloc plus restreint qu'une fonction.

Donnez-leur le même nom mais des valeurs différentes et observez à laquelle on accède dans différentes parties du programme.

**Exercice 32** Ecrivez la fonction **rot** qui prend en argument un caractère et un entier et qui fait une rotation de chaque lettre de la chaîne par l'entier fourni. Exemples : 'a', 1 -> 'b'; 'z', 3 -> 'c'; 'a', 36 -> 'k'.

**Exercice 33** Créez un programme avec deux modules : main et etudiant. Le module etudiant (etudiant.c et etudiant.h) contient les définitions suivantes :

- Le type **etudiant**;
- La fonction **afficher_etudiant**;
- La fonction **creer_etudiant**.

Créez un étudiant dans la fonction *main*.

**Exercice 34** Créez un tableau de 10 étudiants dans la fonction *main*.

**Exercice 35** Ecrivez le programme **creer_classe** qui prend un argument (un nombre entier *naturel*, appelons-le $n$) et qui affiche une classe de $n$ étudiants aléatoires.

**Exercice 36** Ecrivez une fonction qui s'appelle **generer_nom** et qui génère un nom aléatoire.

**Exercice 37** Modifiez votre type etudiant : donnez-leur un nom. Modifiez les fonctions associées.

**Exercice 38** Intégrez tout ça dans dans le programme **creer_classe** (qui fait la même chose qu'avant mais avec des noms en plus). En passant, remarquez que vous avez oublié de modifier la fonction de désallocation (ça m'étonne pas); faites-le puis vérifiez que tout va bien avec valgrind.

**Exercice 39** Pour récompenser les courageux : la première ou le premier qui a terminé *tous* les exercices *et* qui trouve d'où vient l'extrait suivant "mgoaybfaudpqxadxmdfoagxqpmzexqehquzqe" – oui, c'est chiffré, non, c'est pas très difficile – se verra attribuer un **diplôme d'honneur** et offrir une authentique **récompense non-négligeable** sauf s'il sèche cours, TD, ou TP sans raison valable.