**IN 102 - Examen**
30 janvier 2018
François Pessaux

# Énoncé

ENSTA ParisTech

— **Lisez attentivement les consignes et tout le sujet avant de commencer.**

— Les documents (polys, transparents, TDs, livres ...) sont autorisés.
— Vous avez bien entendu accès à la commande `man` qui permet d'obtenir des informations sur les commandes Unix, mais aussi les fonctions de la bibliothèque standard de C (section 3 – ex : `man 3 printf`).
— Sont **absolument interdits** : le WEB, le courrier électronique, les messageries diverses et variées, le répertoire des camarades, le téléphone (même pour avoir l'heure puisque vous l'avez sur votre ordinateur).

— Votre travail sera (en partie) évalué par un mécanisme automatique. Vous **devez** respecter les règles de **nommage** des fichiers et autres **consignes** qui vous sont données.
— **Sauf indications contraires**, vos programmes doivent **gérer** les cas **d'erreur** pouvant survenir. Il vous est demandé de **respecter** la convention C concernant la **valeur retournée** par vos `main`'s ($0 \equiv$ OK, $\neq 0 \equiv$ KO).
— Lorsqu'il vous est demandé que votre programme réponde en affichant «**Yes** » ou «**No** », il ne doit **rien** afficher d'autre, et **pas** «Oui » ou «Yes. » ou «no » ou «La réponse est : no ».
Seuls les **messages d'erreurs** sont autorisés en plus et leur contenu est libre. Donc pensez à retirer vos affichages de test / debug.
— **Indentez** votre code afin que sa lecture ne soit pas un calvaire pour le correcteur ! En `Python` vous y étiez techniquement obligés, en C vous y êtes moralement obligés. La **lisibilité** de vos programmes sera **prise en compte** dans l'évaluation.

— **À la fin de l'examen**, vous devrez créer une **archive** contenant **tous** les fichiers **sources** que vous avez écrits (`.c`, `.h`). Le nom de cette archive devra avoir la structure suivante :
`nom_prenom.zip` ou `.tgz` (selon l'outil d'archivage que vous utilisez).
Par exemple, Indiana Jones nommera son archive `jones_indiana.zip`.
— Vous devrez **copier** cette archive dans répertoire de rendu se trouvant à `~pessaux/in102rendus/`
Par exemple, l'archéologue ci-dessus remettra son examen en invoquant la commande : `cp -vi jones_indiana.zip ~pessaux/in102rendus/`.
— **N'oubliez pas** d'effectuer cette copie sinon nous devrons considérer que vous n'avez rien rendu !

— Le sujet comporte **5** pages et l'examen dure **3** heures.
— Le barème est **volontairement** approximatif.

# 1 Comptage du nombre de bits à 0 ($\sim 15\%$)

Écrivez un programme qui prend en **ligne de commande** un argument représentant une valeur entière (en décimal) et **affiche** en retour le nombre de bits **à 0** dans l'écriture **binaire** de cette valeur.

La valeur issue de la ligne de commande sera stockée sur un **int**.

**Nommage** : Le fichier source de ce programme devra s'appeler `zbits.c`.

**Format de sortie** : Uniquement le nombre de bits à 0 **suivi d'un retour à la ligne**.

**Ex. tests** :

- `./zbits.x 0` ⟶ 32↵
- `./zbits.x 1` ⟶ 31↵
- `./zbits.x 4` ⟶ 31↵
- `./zbits.x 68` ⟶ 30↵
- `./zbits.x 789` ⟶ 27↵
- `./zbits.x` ⟶ «Error. Usage : zbits.x number. »↵

## 2 Comparaison de 3 chaînes de caractères ($\sim 20\%$)

Écrivez un programme qui prend en **ligne de commande** 3 arguments représentant des chaînes de caractères et détermine si ces 3 chaînes sont identiques.

**Attention** : Vous **n'avez pas** le droit d'utiliser la fonction `strcmp ()` de la bibliothèque standard de C.

**Nommage** : Le fichier source de ce programme devra s'appeler `strcmp3.c`.

**Format de sortie** : Uniquement « Yes » (si les 3 chaînes sont **identiques**) ou « No » (si **au moins l'une** des chaînes est **différente** des autres), **suivi d'un retour à la ligne**.

**Ex. tests** :

- `./strcmp3.x Kevin Kevin Kevin` ⟶ Yes↵
- `./strcmp3.x Kevin Kevin Stuart` ⟶ No↵
- `./strcmp3.x Kevin Bob Stuart` ⟶ No↵

## 3 Intersection d'intervalles fermés ($\sim 30\%$)

On donne la structure suivante (dans le fichier `interv.h`), permettant de représenter des intervalles **toujours fermés** sur les **entiers**.

```
------------- interv.h -------------
```

```
#ifndef __INTERV_H__
#define __INTERV_H__

struct interval_t {
  int inf ;
  int sup ;
};

#endif
```

Par **convention**, la borne `inf` est **toujours inférieure ou égale** à la borne `sup`. Un intervalle ne respectant pas cet invariant sera un intervalle « vide » ou « erreur ».

Écrivez un programme qui permet d'entrer au clavier (**pas** via la ligne de commande) 2 intervalles, de calculer leur **intersection** et d'afficher cette intersection.

**Nommage** : Le fichier source de ce programme devra s'appeler `interv.c`.

**Format d'entrée** : un intervalle sera entré au clavier comme sa borne `inf` puis sa borne `sup`.

**Format de sortie** : Uniquement l'intervalle résultant de l'intersection ou « Empty » si l'intersection est vide, le tout **suivi d'un retour à la ligne**. Un intervalle doit être affiché entre crochets **de la manière suivante** : `[`$x$ `;` $y$`]`.

**Ex. tests** :

- `./interv.x`↵
  `3 5`↵
  `4 8`↵
  ⟶ `[4 ; 5]`↵
- `./interv.x`↵
  `1 3`↵
  `6 8`↵
  ⟶ Empty↵

# 4 Matrice ($\sim 35\%$)

Écrivez un programme qui prend en **ligne de commande** 1 argument représentant la taille d'une matrice **carrée** d'entiers à créer. Une fois cette matrice **créée**, il faudra la remplir de la façon suivante :

$$\begin{cases} \forall x,\ t[x][0] = x \\ \forall x\ \forall y,\ y \neq 0 \Rightarrow t[x][y] = t[x][y-1] + 1 \end{cases}$$

Une fois remplie, cette matrice devra être affichée dans le terminal.

**Attention** : La mémoire allouée devra être intégralement et explicitement **libérée** en fin de programme. On ne vous demande **pas** de la libérer dans les **cas d'erreur** (vous pouvez le faire si vous le voulez).

**Nommage** : Le fichier source de ce programme devra s'appeler `mat.c`.

**Format de sortie** : Chaque ligne de la matrice doit être affichée avec un retour à la ligne final. Sur chaque ligne, les valeurs doivent être séparées par **un espace** et la dernière valeur **ne doit pas être** suivie d'un espace puisqu'un retour à la ligne terminera cette ligne de la matrice (regardez l'exemple de sortie ci-dessous).

**Ex. tests** :

- `./mat.x -158` ⟶ ↵
- `./mat.x 5` ⟶
  0␣1␣2␣3␣4↵
  1␣2␣3␣4␣5↵
  2␣3␣4␣5␣6↵
  3␣4␣5␣6␣7↵
  4␣5␣6␣7␣8↵

## —— Fin du sujet ——