IN 101 - Examen
12 janvier 2021
François Pessaux

Énoncé

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

— Les documents (polys, transparents, TDs, livres ...) sont autorisés.
— 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.
— La connaissance de C est un prérequis d'IN101. Ainsi, la présence d'avertissement(s) **réduit la note de 2 pts** (sur 20) de l'exercice considéré, un programme ne compilant pas aura une note **bornée à 10 pts**.
— **Sauf indications contraires**, vos programmes doivent **gérer** les cas **d'erreur** pouvant survenir.
— 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 ». Donc, pensez à retirer vos affichages de test / debug.
— La **lisibilité** et l'efficacité / simplicité / complexité de vos programmes seront **prises 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 :
num`_nom_prenom.zip` ou `.tgz` (selon l'outil d'archivage que vous utilisez) où *num* sera un numéro qui vous sera **attribué personnellement**.
Par exemple, Mickey Mouse nommera son archive `425_mouse_mickey.zip`.
— Vous devrez **copier** cette archive dans répertoire de rendu se trouvant à `~pessaux/in101rendus/`
Par exemple, la souris ci-dessus remettra son examen en invoquant la commande : `cp -vi 425_mouse_mickey.zip ~pessaux/in101rendus/`.
— **N'oubliez pas** d'effectuer cette copie sinon nous devrons considérer que vous n'avez rien rendu !

— Le sujet comporte **3** pages et l'examen dure **2** heures.
— Le barème est **volontairement** approximatif.

1 Bingo ($\sim 60\%$)

On souhaite réaliser un programme qui vérifie si une grille de bingo est gagnante en fonction d'une séquence de nombres.

Le bingo est un jeu composé d'une grille **carrée** remplie de nombres (des entiers) **tous différents**. Des nombres sont alors tirés au hasard tour à tour. À chaque tour, si un joueur trouve dans sa grille le nombre tiré, il le barre. Le gagnant est le premier à avoir barré une ligne, une colonne ou une diagonale de sa grille.

Un jeu sera représenté sous forme de texte dans un fichier contenant :

1. la taille de la grille,
2. les entiers de la grille,
3. les entiers « tirés au hasard ».

Exemple `bing2.txt`

```
4

3 15 2 11
1 4 0 12
7 6 10 13
14 9 8 5

20 20 3 4 10 5 20 20 20
```

Cette grille de 4×4 est gagnante au tour 6, avec la diagonale descendante haut-gauche/bas-droite remplie (dans les nombres tirés au hasard, on a mis des 20 qui n'apparaissent pas dans la grille pour mieux rendre visibles les nombres qui permettent de faire gagner).

Exemple `bing4.txt`

```
4

3 15 2 11
1 4 0 12
7 6 10 13
14 9 8 5

20 20 9 6 20 20 20 20 15 20 20 20
```

Cette grille de 4×4 est perdante car la suite tirée au hasard n'a pas permis de compléter une ligne, colonne ou diagonale.

Écrivez un programme permettant de lire un jeu depuis un fichier texte et de vérifier si la grille est gagnante. Si oui, il faudra dire au bout de combien de tours (les tours sont numérotés en commençant à 1). Le programme prendra le nom du fichier en **argument de ligne de commande**.

Attention : On ne vous demande **pas** de gérer les erreurs dues à un format de fichier incorrect.

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

Format de sortie :

- Si la grille est gagnante le programme devra **afficher** la chaîne `Bingo`, suivie d'un espace suivi du numéro du tour gagnant, terminé par un retour à la ligne. Par exemple :
  `Bingo␣6↵`
- Si la grille est perdante le programme devra **afficher** la chaîne `Ko`, terminée par un retour à la ligne. Par exemple :
  `Ko↵`

Les éventuels messages d'erreur sont libres.

Jeux de données : Quatre jeux de tests vous sont donnés à titre d'exemples : `bing1.txt` (→ "Bingo 5"), `bing2.txt` (→ "Bingo 6"), `bing3.txt` (→ "Bingo 10"), `bing4.txt` (→ "Ko"). Vous pouvez y accéder en téléchargeant l'archive se trouvant à :
`https://perso.ensta-paris.fr/~pessaux/images/foo/abc.tgz`
ou en copiant directement le fichier : `cp ~pessaux/public_html/images/foo/abc.tgz .`

2 Entiers « carrés » (∼ 40%)

On souhaite déterminer si un entier est composé de chiffres formant une suite de nombres dont chacun est le carré du précédent lors de la lecture de son écriture décimale de **gauche à droite**. Si tel est le cas alors on dira (faute de mieux) que l'entier est « carré » et la vérification devra « dire » `Ok`, sinon `Ko`. On commence la vérification toujours avec **un seul** chiffre, le plus à **gauche**.

Exemples : 2416 est bien « carré » car il est formé de 2, 4, 16, chacun étant le carré du précédent (sauf l'initial 2 bien entendu). Il en est de même avec 525625 qui est formé de 5, 25, 625.

Par contre, 2415 n'est pas « carré » car il n'est pas possible d'y retrouver une suite de chiffres format des nombres dont chacun est le carré du précèdent. On a bien 2, 4 mais ensuite 15 n'est pas le carré de 4.

De même, puisque « on commence la vérification toujours avec **un seul** chiffre, le plus à **gauche** », 16256 n'est pas carré car on a 1 suivi de 6 et 6 n'est pas le carré de 1. Il ne faut pas se dire que l'on devrait interpréter ce nombre comme 16 suivi de 256 (le second est le carré du précédent). Vouloir gérer de tels cas compliquerait énormément le programme et ça ne vous est **pas** demandé.

Votre programme prendra l'entier en **argument de ligne de commande**.

Attention : Les carrés grandissant rapidement, pour éviter que les entiers débordent rapidement, vous travaillerez sur des `long int` (et non de simples `int`). Cela vous permettra de faire des tests avec des suites plus longues. Pour rappel, sur les architectures 64 bits, des `long int` peuvent représenter des valeurs entre -9223372036854775808 et 9223372036854775807 alors que des `int` sont limités entre -2147483648 et 2147483647.

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

Format de sortie : **uniquement** la chaîne `Ok` si le test est positif, `Ko` s'il est négatif, le tout terminé par un **retour à la ligne**. Les éventuels messages d'erreur sont libres.

Ex. tests :

- `./sqseq.x 525625` ⟶ `Ok`↵
- `./sqseq.x 12345` ⟶ `Ko`↵
- `./sqseq.x 3981656143046721` ⟶ `Ok`↵
- `./sqseq.x 3981656143046722` ⟶ `Ko`↵
- `./sqseq.x 2416000` ⟶ `Ko`↵

Attention : En testant, ne vous faites pas berner en essayant avec une entrée comme
`./sqseq.x 02416`
qui va vous répondre `Ok` alors que vous pourriez penser que ça devrait répondre `Ko` (2 n'est pas le carré de 0). C'est normal, l'argument de ligne de commande `"02416"` que vous entrez est une chaîne dans le terminal et correspond à **l'entier** 2416. Donc votre fonction de vérification va bien recevoir cet entier 2416.

—— Fin du sujet ——