IN 101 - Examen
10 janvier 2023
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 pré-requis d'IN101. Ainsi, la présence d'avertissement(s) ou d'erreur(s) à la compilation réduira mécaniquement la note de l'exercice considéré.
- **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`). **N'incluez pas** d'exécutables dans l'archive, le mail pourrait la considérer comme un attachement dangereux et le supprimer. Le nom de cette archive devra avoir la structure suivante :
  `nom_prenom.zip` ou `.tgz` (selon l'outil d'archivage que vous utilisez).
- Vous devrez **m'envoyer** cette archive par mail. En cas d'envoi incorrect, il vous sera demandé de refaire l'archive et l'envoi. Par contre, vous ne devrez **surtout pas modifier** les fichiers : leurs dates de dernière modification ne devra pas être ultérieure à l'heure de fin de l'épreuve sous peine d'être considérés comme nuls.
- **N'oubliez pas** d'effectuer cet envoi sinon nous devrons considérer que vous n'avez rien rendu !

- Le sujet comporte 5 pages et l'examen dure **2h30**.
- Le barème est **volontairement** approximatif.

1 Jouer à la bataille simplifiée ($\sim 60\%$)

But macroscopique du programme : On souhaite réaliser un programme qui simule une partie de jeu de cartes dérivée de la bataille. La distribution des cartes aux deux joueurs sera obtenue via un fichier texte. Le programme jouera la partie selon les règles ci-dessous et déterminera les scores et le gagnant s'il n'y a pas match nul.

Règles du jeu : Le jeu se joue à deux, chaque joueur recevant la moitié du paquet de cartes initial (qui aura été mélangé au préalable). Les cartes sont distribuées en nombre égal, une par une, en alternant entre chaque joueur et en commençant par le premier joueur.

À chaque tour, chaque joueur « pose » une carte. Celui qui a la carte visible la plus forte gagne 1 point. Les cartes posées sont ensuite éliminées du jeu.

En cas de cartes égales (situation nommée « *bataille !* ») chaque joueur pose sur sa pile une carte face cachée puis une autre carte face visible. Celui qui a la carte visible (i.e. celle au-dessus de la pile) la plus haute remporte alors 3 points (puisque 3 cartes auront été posées). Si les deux cartes du dessus sont encore égales (encore une « *bataille !* »), le même processus recommence.

La fin de la partie se produit lorsque les joueurs n'ont plus de cartes.

Exemple de configuration : L'état initial du paquet de cartes à distribuer (et donc déjà mélangé) est donné dans un fichier texte contenant :

1. le nombre de cartes du paquet (*size*),
2. *size* entiers représentant les valeurs des cartes.

Pour simplifier, on considérera que la valeur d'une carte est un entier **sans contraintes** (en particulier, on ne cherche pas à respecter les valeurs habituelles As, 2, 3, $\cdots$, Dame, Roi).

Exemple : `btest_3_2.dat`

```
10

1
3
2
2
3
55
4
1
5
10
```

Ce fichier décrit une partie à jouer à partir d'un jeu de 10 cartes dont les valeurs sont 1, 3, 2, 2, etc.

Jeux de données : Des jeux de test vous sont donnés à titre d'exemples, dont le suffixe représente les scores (`btest_`joueur$_1$`_`joueur$_2$`.dat`). Ainsi `btest_3_2.dat` est une distribution de cartes faisant gagner le premier joueur avec un score de 3 contre 2.

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

Format d'entrée : Votre programme prendra en argument de **ligne de commande** le nom du fichier.

Format de sortie :

1. « Scores » suivi d'un espace, suivi du score du premier joueur, suivi d'un espace, suivi du score du second joueur, suivi d'un retour à la ligne final.
2. Le statut du match, suivi d'un retour à la ligne final, ce statut pouvant être :
   - « Match nul »

- « Joueur 1 gagne »
- « Joueur 2 gagne »

Les éventuels messages d'erreur sont libres.

Réponses aux questions de réflexion intermédiaires : Elles devront apparaître en commentaire, numérotées, en début de votre fichier source.

Si vous souhaitez transmettre des photos de schémas, de brouillons, vous serez autorisés à nous envoyer après l'examen ces images, dans la limite **d'une heure après la fin de l'épreuve**. Cela vous laissera le temps d'extraire les photos de votre téléphone et de les envoyer par mail. Merci de ne faire **qu'un seul mail** pour ces potentiels documents.

Question 1.1

Identifiez les grandes étapes du programme, sans descendre dans le détail des fonctions que vous allez écrire. Ce travail représente en quelque sorte l'esquisse de votre futur `main`.

Question 1.2

On s'intéresse à la fonction `load` qui va s'occuper de charger les cartes depuis un fichier et d'initialiser le jeu. Quels sont ses domaines d'entrée et de sortie ?

Question 1.3

En fonction de vos choix à la question précédente, comment identifierez-vous que le chargement des cartes a échoué ?

Question 1.4

Dans quels cas le chargement des cartes peut-il échouer ?

Question 1.5

Esquissez l'algorithme de la fonction `load` qui effectue le chargement du fichier contenant les cartes et retourne le « matériel » pour jouer la partie.

Question 1.6

On s'intéresse maintenant à la fonction `play` qui va jouer la partie et retourner l'état final de celle-ci (i.e. les informations demandées dans la sortie finale, qui seront affichées par la fonction principale – votre `main`). Quels sont les domaines d'entrée et de sortie de `play` ?

Question 1.7

Esquissez l'algorithme de la fonction `play` qui déroule la partie et calcule les scores finaux.

Question 1.8

Que pouvez-vous dire de la relation entre les scores et le nombre de cartes initial du jeu ?

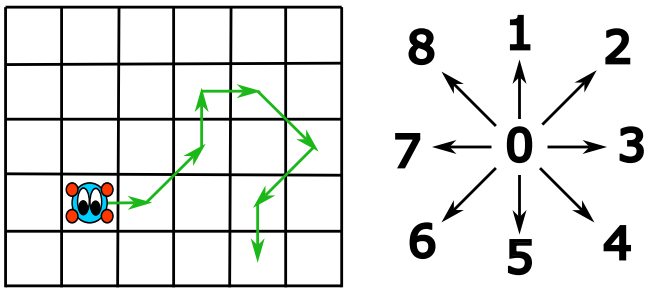
Question 1.9

Transformez votre réflexion algorithmique en un programme C effectif. Respectez les formats d'entrée et de sortie spécifiés en introduction de cet exercice.

2 Robot (~ 40%)

Note : Volontairement, cet exercice n'est pas guidé comme l'est le premier. C'est à vous d'exercer vos compétences de décomposition du problème en sous-problèmes plus simples pour concevoir l'algorithme puis le programme répondant au problème posé.

Un robot (de piètre technologie) se déplace case par case selon une grille. Chaque déplacement élémentaire est donné par un chiffre indiquant une direction selon le codage représenté par le schéma ci-dessous. Dans cet exemple, le robot a suivi le programme 3213465.



Le robot (fonction `robot`) reçoit son programme sous forme d'un **entier** (oui, il est très simple et ne connaît même pas les chaînes de caractères). Il effectue les déplacements indiqués par le programme et doit finalement **retourner** une double indication disant si à l'issue de sa promenade :

- il est revenu à son point de départ ou bien s'il est perdu (donc pas revenu à son point de départ),
- le nombre cases où il s'est effectivement déplacé.

Dans notre exemple, il est perdu après avoir parcouru 7 cases.

Bien évidemment, un opérateur hautement qualifié est chargé, d'une `main`, de fournir au `robot` son programme et d'afficher publiquement les résultats de la mission.

Format d'entrée : Votre programme prendra en argument de **ligne de commande** l'entier servant de programme au robot.

Format de sortie :

1. Si le robot est revenu à son point de départ, afficher la chaîne « De retour en x cases » suivie d'un retour à la ligne final.
2. Sinon, afficher la chaîne « Perdu en x cases » suivie d'un retour à la ligne final.

où x est le nombre de cases calculé. Les éventuels messages d'erreur sont libres.

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

Question 2.1

Écrivez le programme C répondant à ce problème. Vous décrirez tout ce que vous jugerez pertinent comme éléments d'analyse en commentaire au début de votre fichier source. Respectez les formats d'entrée et de sortie spécifiés en introduction de cet exercice.

Indications :

- Afin de pouvoir exécuter des « programmes » de longueur raisonnable et éviter des débordements d'entiers trop rapides, il vous est demandé de travailler non pas avec des `int` mais avec des `unsigned long int` (ce sont simplement des entiers non signés sur 64 bits).

- Le plus grand nombre représentable sur un `unsigned long int` est 18446744073709551615. Donc ne cherchez pas à tester votre programme avec un nombre supérieur. On **ne** vous demande **pas** de vérifier que l'argument de ligne de commande représente bien un entier **ni** qu'il est inférieur ou égal à cette borne.
- La fonction qui permet de convertir la chaîne argument de ligne de commande en un `unsigned long int` est `strtoul` dont la déclaration suit :

  ```
  unsigned long strtoul (char *str, char **endptr, int base) ;
  ```

  où `str` est la chaîne à convertir, `endptr` est à ignorer pour vous et doit être mis à `NULL` et `base` est la base de numération selon laquelle interpréter la chaîne. Cette fonction nécessite `#include <stdlib.h>`.
- Pour afficher, avec la fonction `printf` un `unsigned long int`, l'indicateur de format est `%lu`.

Exemples de tests :

- `./robot.x 3213465` ⟶ `Perdu en 7 cases`↵
- `./robot.x 111` ⟶ `Perdu en 3 cases`↵
- `./robot.x 613` ⟶ `De retour en 3 cases`↵
- `./robot.x 817555232477` ⟶ `De retour en 12 cases`↵

—— Fin du sujet ——