IN 102 - Examen
17 décembre 2019
Goran Frehse

Énoncé

ENSTA ParisTech

Informations générales :
- Le sujet comporte **5** pages et l'examen dure **2** heures.
- Le barème est **volontairement** approximatif.

Autorisations :
- Les documents (polys, transparents, TDs, livres . . .) sont autorisés.
- Sont **absolument interdits :** le web, le courrier électronique, les messageries, les répertoires des camarades, le téléphone (même pour avoir l'heure puisque vous l'avez sur votre ordinateur).

Nommage et Formatage :
- Votre travail sera en partie évalué par un mécanisme automatique. Vous devez respecter les règles de **nommage** des fichiers et des fonctions qui vous sont données. Le programme ne doit **afficher uniquement ce qui est explicitement demandé**, en respectant exactement le format (espaces, virgules, etc.).
- Dans les indications de format, le symbole ␣ représente un espace, ↵ représente un retour à la ligne.
- **En cas de non respect, la note peut être diminuée de 2 points**.

Commentaires :
- Si vos programmes contiennent des commentaires qui clarifient la structure et l'intention du code, **vous pouvez bénéficier d'un bonus de maximum 2 points**. Vous pouvez commenter même si le code est incomplet ou incorrect.

Soumission :
- **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 le répertoire de rendu se trouvant à `~frehse/in102rendus/`
  Par exemple, l'archéologue ci-dessus remettra son examen en invoquant la commande : `cp -vi jones_indiana.zip ~frehse/in102rendus/`

1 Comptage de lettre ($\sim 25\%$)

Écrivez un programme qui prend en ligne de commande un argument représentant une chaîne de caractères (un seul mot, sans espace) ainsi qu'une seule lettre. Ensuite, le programme affiche le nombre de fois que cette lettre apparaît dans la chaîne.

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

Format de sortie : Uniquement le nombre d'occurrences, suivi par un retour à la ligne. En cas d'argument absent, trop d'arguments ou de mauvais argument on affichera le message "Erreur. Usage : lettre.x mot c.".

Ex. tests :

- `./lettre.x oiseau a` ⟶ `1`↵
- `./lettre.x borchtchs a` ⟶ `0`↵
- `./lettre.x anticonstitutionnellement n` ⟶ `5`↵
- `./lettre.x` ⟶ Erreur.␣Usage␣:␣lettre.x␣mot␣c.↵
- `./lettre.x deux mots` ⟶ Erreur.␣Usage␣:␣lettre.x␣mot␣c.↵
- `./lettre.x trois a b` ⟶ Erreur.␣Usage␣:␣lettre.x␣mot␣c.↵

2 Coordonnées en 2D ($\sim 25\%$)

Écrivez un programme dont l'objectif est de calculer les coordonnées 2D d'un point après son déplacement. Les coordonnées du point sont représentés par un `struct`. Chaque coordonnée est constituée par deux entiers, appelés x et y.

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

Format de sortie : Le format d'affichage d'une coordonnée est $(x; y)$ (sans espace). La sortie du programme **terminera par un retour à la ligne**.

Ex. tests :
— `./coord.x` $\longrightarrow (-1; 10)(-2; 7)$ ↵

Q1 Définissez dans votre programme un `struct` pour représenter une coordonnée. Dans votre fonction main, définissez deux coordonnées : $a = (1; 3)$ et $b = (-2; 7)$.

Q2 Écrivez une fonction `afficher` qui prend en argument une coordonnée et l'affiche (sans retour à la ligne).

Q3 Écrivez une fonction `void deplacer` qui prend en argument deux coordonnées c et d (ou des pointeurs) et **modifie** la coordonnée c en additionnant les valeurs de la coordonnée d : $c \leftarrow c + d$.
Attention : Cette fonction n'a pas de valeur de retour.

Q4 Dans la fonction `main` de votre programme, appelez `deplacer` pour additionner b à la coordonnée a. Ensuite, affichez a et b.

3 Morpion ($\sim 50\%$)

Écrivez un programme pour jouer le jeux de morpion entre un humain et l'ordinateur : sur une grille de taille 3×3, chaque joueur place à son tour un symbole (x pour le joueur humain, o pour le joueur ordinateur). Un joueur gagne dès qu'il a placé 3 symboles coté à coté, horizontalement, verticalement ou en diagonale. Le joueur humain commence la partie. A chaque tour, le joueur humain entre un chiffre avec le clavier. Ce chiffre indique la case sur laquelle il souhaite placer son symbole. Le nombre de tours est le nombre de fois que le joueur humain a joué. Les cases sont numérotées comme suivant :

123
456
789

Le joueur ordinateur jouera une case aléatoire, en prenant soin de ne pas jouer sur une case déjà occupée.

Attention : Suivre soigneusement les étapes indiquées. Chaque étape rapporte des points, même si vous n'arrivez pas jusqu'au bout. Coder exactement les fonctions demandées. *Si vous ne trouvez pas la solution d'une sous-question, n'hésitez pas à continuer avec la sous-question suivante.*

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

Format de sortie : Avant l'entrée du chiffre, afficher la grille actuelle, suivie par un retour à la ligne, puis le mot "chiffre␣:␣". A la fin de la partie, afficher la grille actuelle, ensuite soit "Gagné en X tours.", où X est le nombre de tours joués, ou "Perdu en X tours.", suivi par un retour à la ligne. La grille est affiché avec les symboles "." (vide), "x" (joueur humain), "o" (joueur ordinateur).

Ex. tests :

```
./morp.x
...
...
...
chiffre␣:␣3
..x
.o.
...
chiffre␣:␣2
.xx
.o.
o..
chiffre␣:␣1
xxx
.o.
o..
Gagné␣en␣3␣tours.
```

```
./morp.x
...
...
...
chiffre␣:␣3
o.x
...
...
chiffre␣:␣5
o.x
ox.
...
chiffre␣:␣9
o.x
ox.
.ox
chiffre␣:␣2
oxx
ox.
oox
Perdu␣en␣4␣tours.
```

Q1 La grille sera représentée par un tableau d'entiers : 0 représentant une case vide, 1 une case occupée par le joueur humain, 2 une case occupée par le joueur ordinateur.

Créer une fonction `int* creer_grille()` qui crée d'abord un tableau dont les cases sont vides. La fonction doit réserver la mémoire avec `malloc` et retourner un pointeur vers ce tableau.

Q2 Écrivez une fonction `void afficher(int* grille)` qui prend en argument le tableau et affiche les cases sous forme de grille. Voir les tests pour un exemple d'affichage.

Q3 Testez les fonctions précédentes dans une fonction `test_affichage()`. **À l'intérieur de cette fonction**, exécutez toutes les actions suivantes : Créez une grille vide, affichez-la. Marquez ensuite la case 4 comme occupé par le joueur humain, 8 comme occupé par le joueur ordinateur. Affichez le résultat.

Q4 Définissez une fonction `void placer(int* grille, int chiffre, int joueur)` qui prend en argument la grille et place le symbole du joueur à la case correspondant au chiffre. Plus précisément, la fonction modifie la case correspondante du tableau `grille` pour y affecter l'entier qui correspond au joueur (ordinateur ou humain). Pour simplifier, on ne vérifiera pas si la case est réellement libre avant de placer le symbole.

Q5 Définissez une fonction `void placer_alea(int* grille, int joueur)` qui choisit un chiffre entre 1 et 9 de façon aléatoire, puis appelle `placer(...)` pour y placer le symbole. Attention : si la coordonnée est telle que la case n'est pas vide, il faut tirer un nouveau chiffre jusqu'à ce qu'on trouve une case vide.

Astuce : Pour tirer un nombre aléatoire entre 0 et $N-1$, vous pouvez faire appel au générateur de nombres aléatoires avec `rand()%N`. Pensez à initialiser le générateur de nombres aléatoires en appelant `srand(time(0));` une fois au début de votre programme. Il faudra inclure `stdlib.h` et `time.h`.

Q6 Ajoutez une fonction `int a_gagne(int* grille, int joueur)` qui donne 1 si le joueur a gagné et 0 sinon.

Q7 Ajoutez à votre `main` une boucle dans laquelle on
- affiche la grille,
- demande au joueur humain d'entrer un chiffre,
- appelle `placer` pour placer le symbole du joueur humain,
- appelle `placer_alea` pour placer le symbole du joueur ordinateur.

La boucle doit terminer quand un des deux joueurs a gagné. Pour l'instant, on peut ignorer le cas d'un match nul.

Rappel : Suivre les indications dans la section "format de sortie".

Q8 Modifiez votre programme afin de détecter quand le jeu se termine en match nul : aucun des joueurs a gagné mais il n'y a plus de case libre. Dans ce cas, afficher "Match nul en X tours.", où X est le nombre de tours joués.

—— Fin du sujet ——