

DS 2

Informatique MP2I

Julien REICHERT

Toutes les questions de programmation sont à résoudre en OCaml sauf mention explicite du contraire.

Questions de cours ou d'application directe du cours

Question de cours 1 : Donner trois zones de la mémoire virtuelle utilisée dans un programme.

Question de cours 2 : Définir au choix la portée ou la durée de vie d'une variable (préciser ce qu'on définit, les notions n'étant pas exactement les mêmes).

Question de cours 3 : Définir les répertoires `.` et `..` (on rappelle qu'on se base sur les systèmes Unix).

Question de cours 4 : Donner la différence entre `>` et `|` dans une ligne de commande. On pourra s'appuyer sur des exemples d'utilisation expliqués.

Question de cours 5 : Définir les préconditions et les postconditions, exemple à l'appui.

Question de cours 6 : Définir la notion de graphe de flot de contrôle.

Question de cours 7 : Expliquer comment prouver la terminaison d'une fonction récursive, avec un petit exemple (ne pas dépasser une demi-douzaine de lignes, ce serait inutile).

Question de cours 8 : Donner la syntaxe pour remplacer le deuxième élément d'un tableau noté `t` et de taille supposée suffisante par la variable `x` supposée du bon type.

Exercices issus des TD et TP

Exercice T1 : Écrire une formule récursive pour dessiner un « escalier » sur `n` lignes en fonction de `n`. Le rendu pour `n` valant 3 figure ci-après.

```
*  
**  
***
```

Exercice T2 : En supposant l'existence d'une fonction de prototype `int difference_jours(jour j1, jour j2)` prenant en argument deux structures de date et renvoyant le nombre de jours entre la première et la deuxième (strictement positif si la deuxième est ultérieure à la première et négatif sinon), écrire en C une fonction permettant de renseigner le jour de la semaine dans une instance au choix de la structure de date et en s'appuyant sur une instance correspondant à la date actuelle.

Rappel sur la structure de date :

```
struct day { int annee; int mois; int jour; int jdls; };  
typedef struct day jour;
```

Le champ `jdls` correspond au jour de la semaine, avec la convention que le lundi est associé à 1 et le dimanche à 7. Il n'est pas forcément initialisé.

Exercice T3 : Écrire une fonction récursive qui prend en entrée une liste et qui retourne son plus grand élément.

Exercice T4 : Écrire une fonction calculant la somme des éléments d'un tableau d'entiers.

Exercices

Exercice 1 : Écrire une fonction prenant en argument un tableau et déterminant si les variations y alternent, c'est-à-dire que le deuxième est strictement supérieur au premier, que le troisième est strictement inférieur au deuxième, que le quatrième est strictement supérieur au troisième, etc. Il est aussi possible que le deuxième soit strictement inférieur au premier pour lancer l'alternance. **La complexité en espace doit être constante !**

Exercice 2 : Même exercice sur une liste.

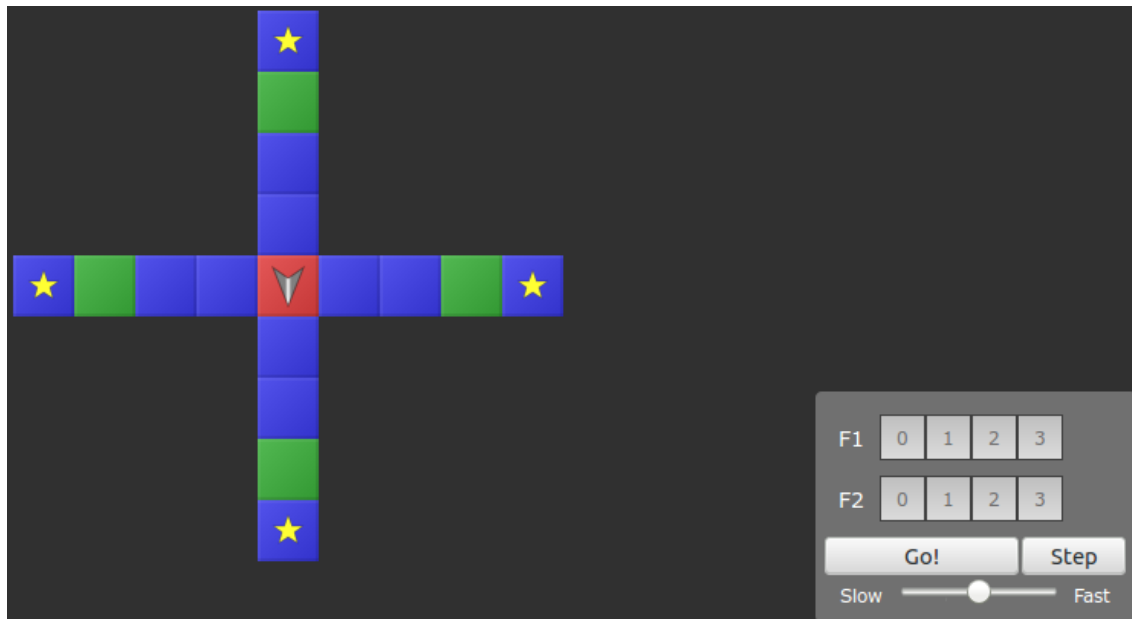
Exercice 3 : Écrire une fonction prenant en argument une chaîne de caractères dont tous les caractères sont des espaces ou des lettres, avec la propriété qu'il n'y a pas deux espaces consécutives ni d'espace en début et en fin de chaîne (cette propriété n'étant pas à vérifier) et renvoyant un quadruplet donnant dans l'ordre le nombre de mots (suite de caractères étant tous des lettres), la plus longue taille d'un mot, l'indice de départ du premier caractère du premier mot le plus long et l'indice de ce même mot dans un tableau qui recenserait les mots si on découpait la chaîne en fonction des espaces.

Exercice 4 : Créer en C une structure de triplets d'entiers dont les champs sont **effectif**, **somme** et **sommecarres**. Écrire alors une fonction prenant en argument un tableau d'entiers et sa taille et construisant une instance de la structure créée et contenant la taille du tableau, la somme des éléments du tableau et la somme des carrés des éléments du tableau.

Problème : Robozzle

Parmi les liens sur mon site figure une mention d'un jeu consistant à « programmer avec des robots pour résoudre des énigmes ». Ce sujet ne se contente pas de donner des énigmes à résoudre comme je l'ai déjà fait par le passé mais à simuler les mécanismes du jeu.

Voici un exemple d'instance du jeu :



On observe une flèche pointant au début vers le bas, des cases de différentes couleurs, certaines possédant une étoile, et un champ en bas à droite pour saisir des commandes associées à des fonctions appelées F1 et F2 (dans le cas général, il peut y avoir plus de fonctions et plus d'emplacements pour ces fonctions).

Le but du jeu est d'amener la flèche à collecter toutes les étoiles en se déplaçant sur les cases sans sortir de la zone de déplacement (un déplacement vers la zone noire fait perdre le jeu).

Les commandes disponibles sont avancer, tourner à gauche, tourner à droite et appeler une fonction. Chacune de ses commandes peut être protégée ou non par un test conditionnel sur la couleur de la case : soit une couleur est imposée sinon la commande est ignorée, soit la commande est exécutée quelle que soit la couleur.

Lorsqu'une fonction est appelée, l'appel actuel est mis en attente (voir le principe de la récursivité et les blocs d'activation). Si l'enchaînement des appels de fonction s'arrête, le jeu s'arrête et il est perdu s'il reste des étoiles à collecter. Au contraire, au moment où la dernière étoile est collectée, le jeu s'arrête immédiatement même s'il reste des appels à traiter, et la partie est gagnée. La partie démarre par un appel de la première fonction.

Faute d'avoir abordé les types enregistrements en OCaml, la simulation du jeu se fera de manière un peu artisanale, et les grilles de jeu seront des tableaux de tableaux contenant des couples formés par un entier (la couleur entre 0 et 3, sachant que 0 représente un trou) et un booléen (`true` si, et seulement si, la case contient une étoile).

Une instance de jeu sera un couple formé d'une grille de jeu et de la position de départ de la flèche, sous forme d'un triplet de références d'entiers (indice de ligne, indice de colonne, orientation), la ligne 0 étant tout en haut, la colonne 0 étant tout à gauche et les orientations de 0 à 3 étant respectivement droite, haut, gauche et bas.

La suite du sujet se fera avec la convention bleu = 1, vert = 2, rouge = 3.

Question P1 : Construire l'instance de jeu donnée en exemple, en se limitant à neuf lignes et neuf colonnes pour la grille de jeu.

Question P2 : Écrire une fonction déterminant si une instance de jeu est correcte, c'est-à-dire qu'aucun élément de la grille correspondant à un trou n'est une case avec une étoile ou la case de départ.

(Les graphes et les parcours n'ayant pas encore été abordés, il est impossible de donner en exercice la vérification qu'il est possible d'aller de la case de départ à toutes les cases ayant une étoile.)

Question P3 : Écrire une fonction déterminant si une instance de jeu est réduite, c'est-à-dire qu'il existe au moins une case qui n'est pas un trou dans la première ligne, dans la dernière ligne, dans la première colonne et dans la dernière colonne.

On supposera désormais que toutes les instances de jeu respecteront les contraintes de ces deux dernières fonctions.

Pour représenter une soumission du joueur, on utilisera des tableaux de tableaux de commandes (les tailles des tableaux de commandes ne seront pas forcément identiques d'une fonction à l'autre, en particulier un emplacement non utilisé ne sera pas dans le tableau), chaque commande étant un couple d'entiers selon la convention suivante :

- Le premier entier vaut -1 pour la commande « aller tout droit ».
- Le premier entier vaut -2 pour la commande « tourner à gauche ».
- Le premier entier vaut -3 pour la commande « tourner à droite ».
- Un premier entier positif déclenche un appel à la fonction correspondante (à l'indice correspondant du tableau de tableaux de commandes, donc).
- Le deuxième entier vaut 0 s'il n'y a pas de contrainte sur la couleur, et sinon il s'agit de la contrainte sur la couleur associée.

Une solution possible (il est censé en exister une avec sept commandes au total) à l'instance donnée en exemple est, avec les notations intuitives :

F1 : F2 F1
 F2 :

Question P4 : Représenter cette solution en OCaml selon les principes ci-avant.

Question P5 : Écrire une fonction `fini` prenant en argument une grille de jeu en cours de partie et déterminant si toutes les étoiles ont été ramassées.

Question P6 : Écrire deux fonctions `gauche` et `droite` prenant en argument une instance et la mutant en faisant l'action correspondante sur la flèche.

Question P7 : Écrire une fonction `avancer` prenant en argument une instance et la mutant en avançant la flèche. Il s'agit de plus de ramasser l'éventuelle étoile rencontrée, déclencher une erreur (`failwith "Game over !"`) si on quitte la zone de jeu, et déclencher une erreur (`failwith "Victoire !"`) si la dernière étoile a été ramassée.

Pour simuler l'exécution du jeu, une possibilité est de convertir les tableaux de commandes en autant de fonctions mutuellement récursives, mais ceci présuppose que le nombre de tableaux de commandes est fixé car du hardcodage est en jeu. Pour autant, la beauté du geste incite à faire cette petite digression.

Question P8 : Écrire une fonction prenant en argument deux tableaux de commandes, représentant les indices 0 et 1, et renvoyant deux fonctions récursives les représentant. On utilisera une variable globale appelée `instance` représentant l'instance de jeu associée.

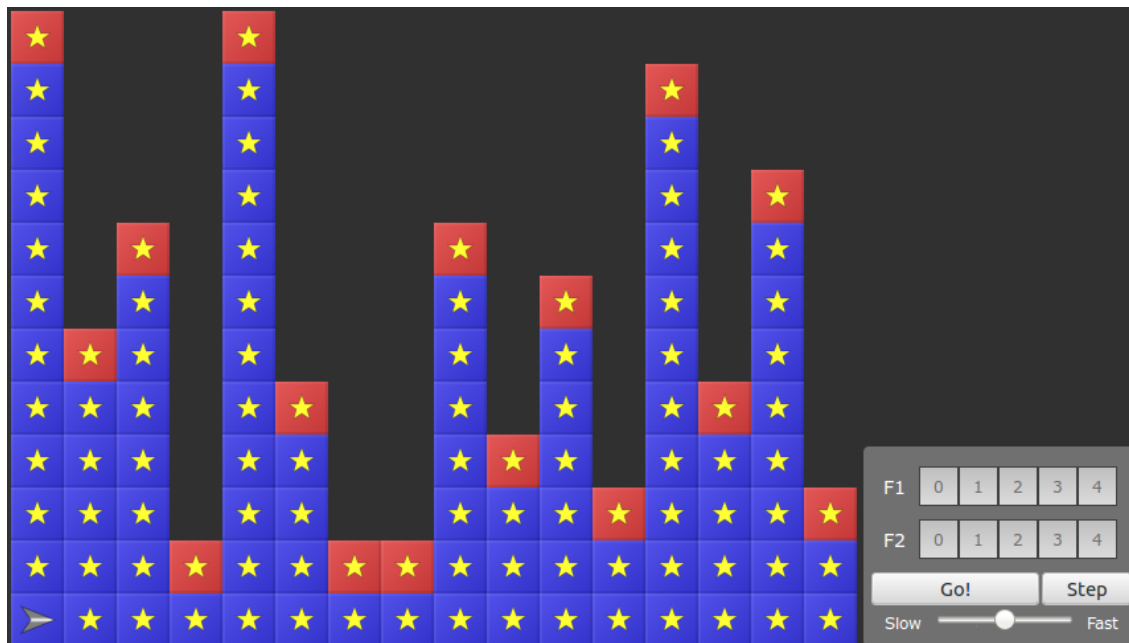
Comme suggéré en introduction de la question précédente, on ne se servira pas de la fonction qui y a été écrite par la suite. Il s'agira de maintenir dans une récursion qui ne s'arrête que sur une fin de jeu une liste des appels à traiter, la tête de liste étant l'appel en cours, et chaque élément de la liste, représentant l'appel, étant un couple d'entiers indiquant l'indice du tableau de commandes au sein du tableau de commandes et l'indice de la prochaine commande à faire dans ce tableau. On reprend alors en version simplifiée l'idée des blocs d'activation.

Question P9 : Quelle est nécessairement la valeur de la liste au début de l'exécution du jeu ?

Question P10 : Écrire une fonction prenant en argument une instance de jeu et renvoyant une copie de celle-ci.

Question P11 : Écrire une fonction prenant en argument une instance de jeu et une soumission de joueur et déterminant si la soumission est correcte. On veillera à ne pas muter la grille de jeu.

Question P12 : Résoudre l'instance du jeu ci-après, ce qui montre une pleine maîtrise des principes de la récursivité. On pourra proposer la solution sous la forme donnée en introduction de la question P4 (le stylo rouge est autorisé, et plus généralement tout encodage dont la compréhension est intuitive est accepté).



Question P13 (bonus) : Trouver une solution en sept commandes au total à l'instance donnée en exemple.