DS 1

Informatique pour tous, deuxième année

Julien Reichert

Durée : Quatre heures.

Ce devoir est composé d'une partie de cours + application et d'un sujet d'annales distribué séparément.

Détail amusant : j'avais prévu de rédiger un exercice consistant à faire un interpréteur de SQL en Python, puis je me suis souvenu qu'un sujet de concours l'avait déjà fait !

Exercice C-1 : Réaliser une implémentation de la structure de pile : expliquer la structure choisie et écrire les trois fonctions de base associées, ainsi que les fonctions `sommet`, `est_vide` et `taille`.

Exercice C-2 : Soit une fonction récursive telle que le coût pour le paramètre $2n$ soit n plus le coût pour le paramètre n, le coût pour le paramètre 1 étant 1. Quelle est la complexité exacte de cette fonction pour le paramètre 2^k, et quel est l'ordre de grandeur pour un paramètre quelconque, si le coût est supposé croissant [1] ?

Exercice A-1 : Sans disposer de mémoire autre que des piles, combien de piles supplémentaires au moins sont nécessaires pour « retourner » une pile de taille $n \geq 2$ [2] ? Combien d'empilements et de dépilements sont alors effectués ? Écrire un programme en Python pour réaliser cette opération, en utilisant l'implémentation et les fonctions de la question de cours et en supposant qu'il n'y a pas de problème de capacité.

Exercice A-2 : Écrire une fonction récursive appliquant l'algorithme d'Euclide. Écrire ensuite une fonction déterminant un couple de Bézout.

1. bonus si un exemple de fonction, même artificiel, est écrit ensuite
2. c'est-à-dire faire en sorte que l'ordre des éléments dans la pile de départ soient inversés

Exercice A-3 : Algorithme shunting-yard [3].

Dans le TP sur les applications des piles, un exercice consistait à faire évaluer à l'aide d'une pile une expression en notation polonaise inversée. L'algorithme à écrire ici revient à réécrire une expression mathématique usuelle en une expression équivalente en notation polonaise inversée.

Le principe est de lire l'entrée découpée en « jetons », c'est-à-dire des unités lexicales, les jetons possibles étant des nombres, des opérateurs et des parenthèses ouvrantes ou fermantes, et de transférer les nombres sur la sortie mais de mettre en attente les opérateurs dans une pile d'opérateurs afin de pouvoir tenir compte des priorités opératoires dans l'expression finale.

Ainsi, l'expression `2 + 3 x 5` donnera `2 3 5 x +`, alors que `(2 + 3) x 5` donnera `2 3 + 5 x`.

Écrire l'algorithme en Python à partir des indications en pseudo-code ci-après. On admettra que les opérateurs se limiteront à l'addition, la multiplication, la soustraction et la division et que les nombres seront limités aux entiers positifs. La sortie sera au choix la liste des jetons dans l'ordre d'apparition dans l'expression équivalente en notation polonaise inversée ou une chaîne de caractères obtenue par la fusion des jetons dans le même ordre (moins pratique pour un traitement ultérieur).

Fonctionnement de l'algorithme restreint aux conditions de l'exercice :

Pour tous les jetons :
- Si c'est un nombre, l'ajouter à la sortie.
- Si c'est un opérateur, dépiler et ajouter à la sortie tous les opérateurs de priorité égale ou supérieure (attention à s'arrêter si on rencontre une parenthèse ouvrante) puis empiler l'opérateur lu. Ainsi, le seul cas sans dépilement est quand un opérateur multiplicatif est lu alors qu'un opérateur additif est dans la pile.
- Si c'est une parenthèse ouvrante, l'empiler.
- Si c'est une parenthèse fermante, dépiler tous les opérateurs jusqu'à la première parenthèse ouvrante rencontrée et les mettre dans la sortie (pas la parenthèse, évidemment).

À la fin, dépiler et ajouter à la sortie tous les opérateurs.

On admet que l'expression ne causera pas d'erreur de syntaxe.

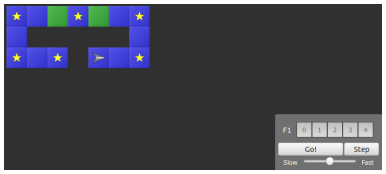
3. Quelle ne fut pas ma surprise de découvrir que ce n'était pas le nom des auteurs !

Exercice A-4 : Robozzle.

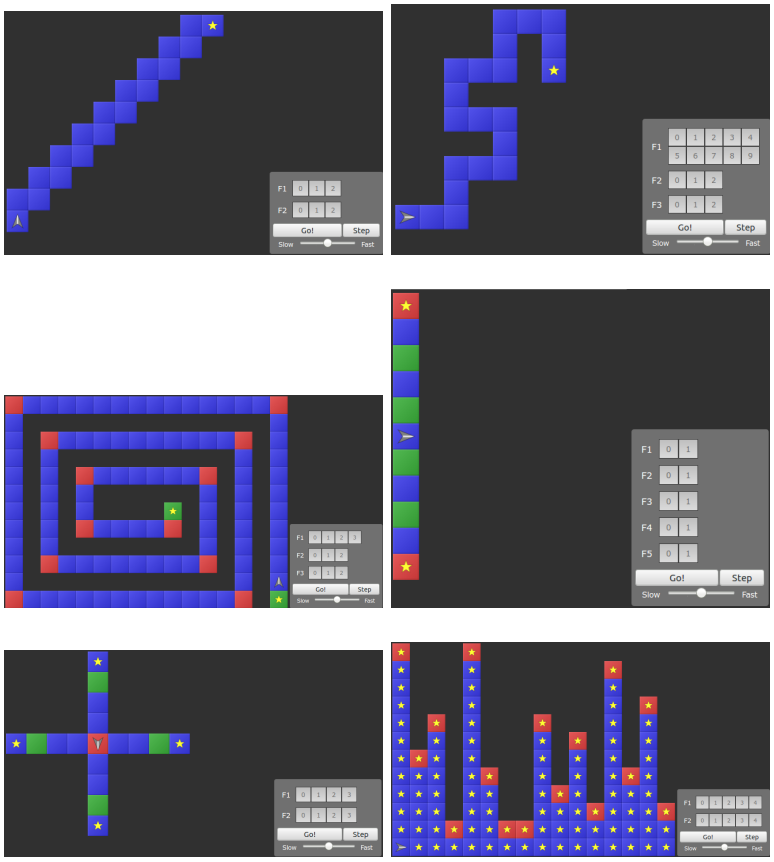
Une instance de Robozzle est un plateau de jeu contenant des cases de trois couleurs différentes : des bleues, des rouges et des vertes, certaines de ces cases étant marquées d'une étoile, d'un robot présent sur une des cases avec une orientation quelconque parmi les quatre directions cardinales, et d'une liste de fonctions à remplir avec pour chaque fonction une limite d'instructions. Les instructions possibles sont de la forme `si condition alors action`, avec 4 conditions possibles : vrai, la case du robot est bleue, la case du robot est rouge, la case du robot est verte. Les actions possibles sont l'avancée d'une case, un virage à gauche sur place, un virage à droite sur place et un appel récursif à n'importe laquelle des fonctions. Une instance est résolue si on parvient à renseigner les fonctions disponibles (pas nécessairement toutes) dans la limite de la place disponible de sorte que le robot passe par toutes les cases marquées d'une étoile (ce qui interrompt immédiatement l'exécution des instructions) sans jamais sortir du plateau (par exemple en avançant à un endroit où il n'y a pas de case). L'exécution doit démarrer sur F1, en outre.

À titre d'exemple, l'instance suivante peut se résoudre en fournissant, avec une fonction de 5 instructions disponible :

F1 : si vrai alors avancer (qu'on écrira simplement « avancer »), avancer, tourner à gauche, si la case du robot est verte alors tourner à droite (qu'on écrira simplement « droite vert ») et répéter F1.



Résoudre les six autres instances du jeu ci-dessous.



Résoudre la dernière instance signifie que les piles et la récursivité sont pleinement maîtrisées. . .