TP 4

Exercice 1 : Manipulation de pile

.1)

P symbolise l'ensemble des piles, N les entiers.

$$\text{EMPTY}\ \frac{}{empty = []}$$

$$\text{PUSH}\ \frac{p \in P, e \in N}{push(p, e) \in P}$$

$$\text{TOP}\ \frac{p \in P}{top(p) \in N}$$

$$\text{POP}\ \frac{p \in P}{pop(p) \in P}$$

.2)

Soit p une pile quelconque, p = [1,2]. L'élément à gauche est en haut de la pile, si on push un 3 dans la pile p, on aura p = [3,1,2].

$$\frac{}{empty \overset{stack}{\Longrightarrow} []}$$

$$\frac{p \overset{stack}{\Longrightarrow} p, e \overset{stack}{\Longrightarrow} e}{push(p, e) \overset{stack}{\Longrightarrow} [e, p]}$$

$$\frac{p \overset{stack}{\Longrightarrow} [e1, e2]}{top(p) \overset{stack}{\Longrightarrow} e1}$$

$$\frac{p \overset{stack}{\Longrightarrow} [e1, e2]}{pop(p) \overset{stack}{\Longrightarrow} [e2]}$$

.3)

Fichier .pl | usage :

empty([]). → retourne TRUE.
top([2,1],R). → retourne R=2.
pop([2,1],R). → retourne R=[1].
push([1],2,R). → retourne R=[2,1].

Exercice 2 : Avec des GOTOs...

.1)

(1 :) empile un 1 → [1]
(1 :) empile un 1 → [1,1]
(2 :) empile un 2 → [2,1,1]
(+ :) consomme les deux valeurs du top, les additionne, et empile le resultat → [3,1]
(+ :) consomme les deux valeurs du top, les additionne, et empile le resultat → [4]
(LABEL=1) pose le label n=1 → [4]
(4 :) empile un 4 → [4,4]
(- :) consomme les deux valeurs du top, les soustrait, et empile le resultat → [0]
(JUMP=1) vérifie que la valeur au top est égal à 0 → true : GOTO LABEL=1 → [0]
(4 :) empile un 4 → [4,0]
(- :) consomme les deux valeurs du top, les soustrait, et empile le resultat → [4]
(JUMP=1) vérifie que la valeur au top est égal à 0 → false : continue → [4]
(5 :) empile un 5 → [5,4]
(+ :) consomme les deux valeurs du top, les additionne, et empile le resultat → [9]

.2)

D'abbord on a les modificateurs (M) qui modifient l'état de la pile, les entiers (I) qui sont les valeurs qu'on met dans la pile, les operateurs (O) qui

sont les opérations qu'on applique aux éléments de la pile, finalement il y a les GOTOs (G) qui permetent de changer l'ordre des opérations, (C) est la commande.

C := M | G
M := O | I
O := + | - | * | /
I := n ∈ N
G := label | jump

.3)

P est l'ensemble des piles.

$$\text{I}\frac{C \in I, p \in P}{push(e,p) \in P}$$

$$\text{O}\frac{C \in O, p \in P, p1 = pop(p), p2 = pop(p1)}{push((top(p))\ O\ (top(p1)), p2) \in P}$$ O est l'opération

$$\text{G}\frac{C \in G, p \in P}{p = p}$$

.4)

$$\frac{p \overset{exo2}{\Longrightarrow} [e1, e2, e3]}{O(p) \overset{exo2}{\Longrightarrow} [e1\ O\ e2, e3]}$$

$$\frac{p \overset{exo2}{\Longrightarrow} [e2], I \overset{exo2}{\Longrightarrow} e1}{I(p) \overset{exo2}{\Longrightarrow} [e1, e2]}$$

$$\frac{p \overset{exo2}{\Longrightarrow} p}{G(p) \overset{exo2}{\Longrightarrow} p}$$

.5)
Fichier .pl | usage :

commandes2(C,P,R). → C est la liste des commandes le deuxième argument represent l'état initial de la pile, on retourne le résultat final dans R.
Une suite de commandes doit être sous forme de liste : [1,2,+] → empile 1, 2, les additionne

commandes2([1,2,+],[],R). → R = [3].
commandes2([1,+],[2],R). → R = [3].
commandes2([1,+],[],R). → false
commandes2([1,1,+,+,[label,1],4,-,[jump,1],+],[],R). → R = [9].
label= 1 → [label,1]

Exercice 3 : Et avec des fonctions ?

.1)
: CARRE DUP *; → doit recevoir comme input une pile → [n]

(DUP :) duplique l'élément au top de la pile → [n,n]
(* :) consomme les deux valeurs du top, les multiplie, et empile le resultat → $[n^2]$

.2)
On a les commandes (C), modificateurs (M) comme dans l'exo 2, ((I) et (O)), et en plus on a les fonctions (F), on n'a plus les GOTOs (G).

C := M
M := O | I | F
O := + | - | * | /
I := n ∈ N
F := swap | dup | over | drop | carre

.3)

: DROP pop(p) ;

$$\text{DROP}\frac{p \in P}{drop(p) \in P}$$

: DUP push(top(p),p) ;

$$\text{DUP}\frac{p \in P}{dup(p) \in P}$$

: OVER push(top(pop(p),push(top(p),pop(p))) ;

$$\text{OVER}\frac{p \in P}{over(p) \in P}$$

: SWAP push(top(pop(p)),push(top(p),pop(pop(p)))) ;

$$\text{SWAP}\frac{p \in P}{swap(p) \in P}$$

: CARRE push(top(p),p) *; = : CARRE dup *;

$$\text{CARRE}\frac{p \in P}{carre(p) \in P}$$

.4)

$$\frac{p \overset{exo3}{\Longrightarrow} [e1, e2, e3]}{drop(p) \overset{exo3}{\Longrightarrow} [e2, e3]}$$

$$\frac{p \overset{exo3}{\Longrightarrow} [e1]}{dup(p) \overset{exo3}{\Longrightarrow} [e1, e1]}$$

$$\frac{p \overset{exo3}{\Longrightarrow} [e1, e2]}{over(p) \overset{exo3}{\Longrightarrow} [e2, e1, e2]}$$

$$\frac{p \overset{exo3}{\Longrightarrow} [e1, e2, e3]}{swap(p) \overset{exo3}{\Longrightarrow} [e2, e1, e3]}$$

$$\frac{p \overset{exo3}{\Longrightarrow} [e1, e2]}{carre(p) \overset{exo3}{\Longrightarrow} [e1^2, e2]}$$

.5)

Fichier .pl | usage :

commandes3(C,P,R). → C est la liste des commandes le deuxième argument represent l'état initial de la pile, on retourne le résultat final dans R.

Une suite de commandes doit être sous forme de liste : [1,2,+] → empile 1, 2, les additionne

commandes3([1,2,+,carre],[],R). → R = [9].
commandes3([1,+,drop],[2],R). → R = [].
commandes3([drop],[],R). → false
commandes3([1,4,carre,swap,drop,dup],[],R). → R = [16,16].

.6)

Ces fonctions sont des Modificateurs (M) de pile, ce qui veut dire que par exemple si drop, appelle recursivement drop, à un moment on va avoir une pile vide, et il y aura une erreur, on peut aussi créer une boucle infinie croissante avec dup, en duplicant toujours la valeur au top.

Exercice 4 : Conditionnelles

.1)

Imaginons l'appele exo2 suivant [1,2,[label,1],4,4,-,[jump,1],[2]] alors le code va boucler entre lable 1 et jump 1, car le top de la pile sera toujours 0 avant jump 1, ceci serait impossible avec l'exemple de l'exerceice 4, car le input sera encadre dans l'un des if, et on pourra pas retourner en arrière.

.2)

On a les commandes (C), modificateurs (M) : ((I) et (O) et (F)), et en plus on (E) la fonction eggsize

C := M
M := O | I | F | E
O := + | - | * | /
I := n ∈ N
F := swap | dup | over | drop | carre
E := eggsize

.3)

.4)

.5)

Fichier .pl | usage :

commandes4([1,2,+,eggsize],[],R). → R= [3,3], et print "reject".