Exercice 1 : Tableaux dynamiques

On appelle tableau dynamique une structure permettant la représentation d'une collection, indexée, de cardinal non borné, d'éléments au moyen de tableaux dont la taille s'adapte à la taille de la collection à représenter.
Définir un type structuré `array_dyn` contenant :
— un entier `alloc` contenant la taille du tableau `ctnts` alloué ;
— un tableau `ctnts` *d'entiers* de taille `alloc` éléments de la collection.

Q. 1 Définir une fonction `empty()` retournant un tableau dynamique vide.

Q. 2 Définir une fonction `void realloc_if_needed(array_dyn d, int i)` qui, au besoin, réalloue le champs `ctnts` de `d` de sorte que la case `i` soit a l'intérieur du tableau `ctnts`. On fera attention à libérer la mémoire inutile.

Q. 3 Définir une fonction `void write(array_dyn d, int i, int x)` écrivant `x` dans la `i`-ième case du tableau `d`. Votre fonction doit fonctionner pour chaque $i \geqslant 0$.

Q. 4 Définir une fonction `int read(array_dyn d, int i)` lisant la `i`-ième case du tableau `d`. Votre fonction doit fonctionner pour chaque $i \geqslant 0$.

Exercice 2 : Piles en OCaml avec des listes

On souhaite représenter une pile en OCaml à l'aide d'une liste. Le haut de la pile est la tête de la liste.

Q. 1 Définir un type `'a pile` représentant une pile contenant des éléments de type `'a`

Q. 2 Définir une fonction `est_vide : 'a pile -> bool` prenant en argument une pile et s'évaluant en un booléen indiquant si la pile est vide.

Q. 3 Définir une fonction `vide : unit -> 'a pile` prenant unit en argument et s'évaluant en une pile vide.

Q. 4 Définir une fonction `empiler : 'a -> 'a pile -> 'a pile` prenant en arguments un élément et une pile et s'évaluant en la pile obtenue après insertion de l'élément dans la pile.

Q. 5 Définir une fonction `depiler : 'a pile -> ('a * 'a pile)` prenant en argument une pile et retournant une paire formée de l'élément qui était en tête de pile et de la nouvelle pile.

Exercice 3 : NPI

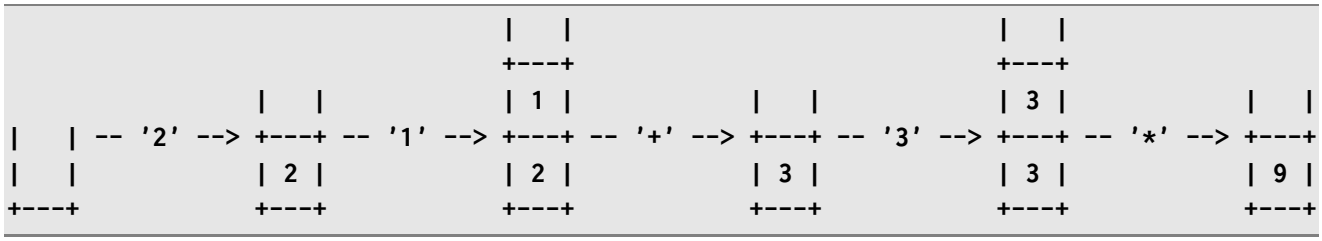
La description classique d'une expression arithmétique a le mauvais goût de nécessiter l'utilisation de parenthèses. Par exemple dans l'expression : $((3\times(2+2))-(7\times2)/((3\times1)+2))$.
Il est toutefois possible de supprimer cette contrainte, en *postfixant* les opérateurs arithmétiques. Aussi plutôt que d'écrire `3*(2+1)` on écrira : `2 1 + 3 *` ce qu'on lit de droite à gauche comme :
— L'application d'une multiplication à :
  — L'application d'une addition à :
    — 2

— 1
— 3

On appelle une telle notation la *notation polonaise inversée* (NPI).

Q. 1 Donner la NPI de l'expression $((3\times(2+2))-(7\times2)/((3\times1)+2))$.

L'évaluation d'une telle expression se fait aisément au moyen d'une pile des opérandes en attente :



Q. 2 Définir un algorithme d'évaluation d'une expression sous forme NPI.

Q. 3 Définir un type OCaml permettant la représentation d'une expression sous forme NPI. On pourra s'aider d'une liste mêlant des entiers et des opérateurs.

Q. 4 Implanter l'algorithme de calcul d'une expression sous forme NPI au moyen d'une pile implantée en OCaml.

Exercice 4 : Piles en C par tableaux dynamiques

Dans cet exercice, on représente une pile en C, par un tableau dynamique. Une pile est représentée par un tableau de ses éléments, le haut de la pile est la dernière case occupée du tableau.

Q. 1 Définir en C le type `pile`.

Q. 2 Implanter les opérations `pile pile_vide();`, `bool est_vide(pile p);`, `void empiler(int elem, pile* p)`, et `int depiler(pile* p)`. Noter que la signature des fonctions induit une manipulation *par effet de bord* de vos piles.

Exercice 5 : Piles en C par maillons chaînés

Dans cet exercice, on représente une pile en C, par une séquence de maillons simplement chaînés.

Q. 1 Définir en C le type `pile`.

Q. 2 Implanter les opérations `pile pile_vide();`, `bool est_vide(pile p);`, `void empiler(int elem, pile* p)`, et `int depiler(pile* p)`. Noter que la signature des fonctions induit une manipulation *par effet de bord* de vos piles.

Exercice 6 : Piles en OCaml avec des fonctions

On souhaite représenter une pile en OCaml à l'aide d'une fonction et d'un entier. Par exemple la pile formée de la paire : `{content = f; size = n}` est la pile contenant (le haut de la pile est en premier) : `(f 0), (f 1), ..., (f (n-1))`

Q. 1 Définir un type `'a pile` représentant une pile contenant des éléments de type `'a`

Q. 2 Définir une fonction `est_vide : 'a pile -> bool` prenant en argument une pile et s'évaluant en un booléen indiquant si la pile est vide.

Q. 3 Définir une fonction `vide : unit -> 'a pile` prenant unit en argument et s'évaluant en une pile vide.

Q. 4 Définir une fonction `empiler : 'a -> 'a pile -> 'a pile` prenant en argument un élément et une pile et s'évaluant en la pile obtenue après insertion de l'élément dans la pile.

Q. 5 Définir une fonction `depiler : 'a pile -> ('a * 'a pile)` prenant en argument une pile et retournant une paire formée de l'élément qui était en tête de pile et de la nouvelle pile.