Université de Paris

L2 informatique - Année 2021–2022

TD d'Éléments d'Algorithmique n° 7

Exercice 1. *Insertion dans une liste doublement chaînée.*
Écrivez le pseudocode d'une fonction INSERER(L, k, x) qui prend comme argument une liste doublement chaînée L, un élément non nul x de L et une clé k et insère un nouveau nœud ayant la clé k avant x.

Rappel. Une pile est une structure de données *abstraite* sur laquelle sont définies trois opérations :

- `empty(P)` qui teste si la pile `P` est vide ;
- `push(x,P)` qui ajoute un élément `x` au sommet de la pile `P` ;
- `pop(P)` qui enlève la valeur au sommet de la pile `P` et la renvoie.

Exercice 2. *Sedgewick.*
Dans la suite suivante, une lettre indique un *push* et un astérisque indique un *pop*. Donnez la suite de valeurs retournées par les *pop* exécutés lorsqu'on applique à une pile la suite d'opérations suivante :

```
L A * S T I * N * F I R * S T * * O U * T * * * * * *
```

Exercice 3. *Réalisations concrètes.*

1. Implémentez une pile en la représentant comme une liste chaînée simple.
2. On suppose maintenant que la taille maximale de la pile est donnée lors de sa création. Implémentez une pile en la représentant par la structure suivante :

```
class Stack {
  int[] T;
  int top;
}
```

Exercice 4. *La récursion revisitée.*

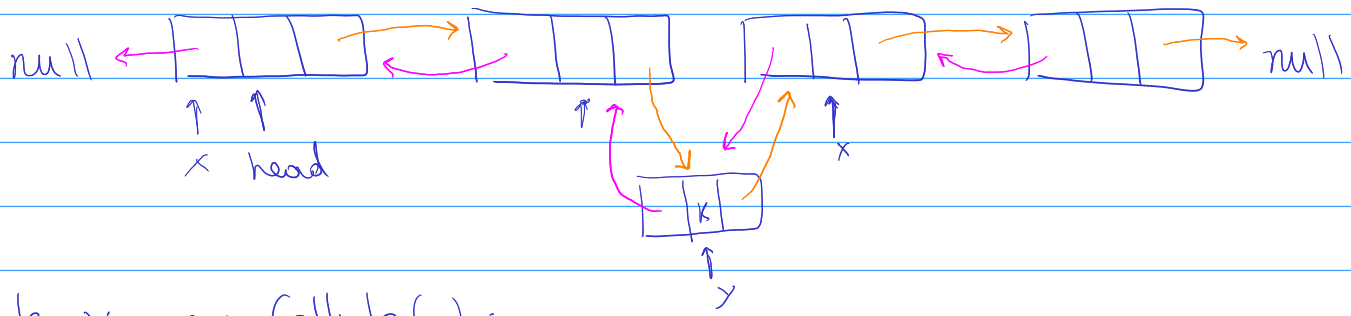
1. Écrivez un algorithme itératif qui, étant donnée une liste `L`, retourne une nouvelle liste qui contient les éléments de `L` dans l'ordre inverse. Votre algorithme s'exécutera en temps et en espace linéaires (proportionnels à la longueur de `L`), et il pourra détruire la liste passée en paramètre.
2. Écrivez un algorithme récursif qui, étant donnée une liste doublement chaînée `L`, retourne une nouvelle liste qui contient les éléments de `L` dans l'ordre inverse. Vous ne devez pas utiliser de mémoire supplémentaire, sauf pour les pointeurs.
3. Écrivez un algorithme récursif qui concatène deux listes. Vous ne devrez utiliser aucun espace en plus de celui qui est utilisé pour implémenter la récursion.
4. Écrivez un algorithme itératif qui concatène deux listes. Vous pouvez détruire l'une des deux listes.

On suppose maintenant données des opérations `read` et `write` qui lisent et écrivent un caractère, et l'opération `tochar` qui convertit un entier en le caractère correspondant.

5. Écrivez un algorithme itératif qui lit un nombre `n` de caractères et les affiche dans l'ordre inverse.
6. Écrivez un algorithme itératif qui affiche la valeur d'un nombre en base 8.

Exo 1 :

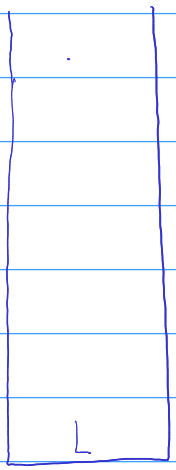
Cellule :	ListeDC :
Objet data; Cellule prev; Cellule next;	Cellule head;




```
Cellule y = new Cellule( );
y.data = x;
if ( x.prev ≠ null) {
    x.prev.next = y;
}
y.next = x
y.prev = x.prev;        (// ça marche même si x.prev = null)
if ( x.prev = null) {
    head = y;
} else {
    x.prev = y;
}
```

Exo 2 :

AINRTSUTOIFTS

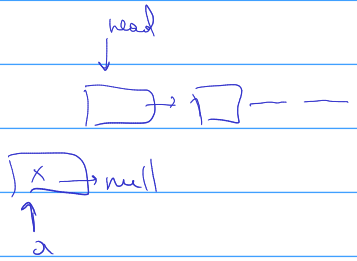


Exo 3 :

Cellule :
Objet data
Cellule next

Liste :
Cellule : head

Pile avec listes :

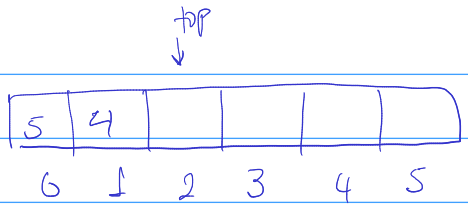


Stack :
Liste l;

```
fonction empty ( ) {
    return (l.head = null);
}
fonction push (Objet x) {
    Cellule a = new Cellule(x, null);
    a.next = l.head;
    head = a;
}
fonction pop( ) {
    if (!.empty) {
        Objet o = l.head.data;
        l.head = l.head.next;
        return o;
    }
    else return null;
}
```

On choisit l'implémentation où push insère un élément à la tête de la liste, pour que push soit constante. (Si on empile des nouveaux éléments à la fin de la liste, l'opération push prendra temps O(n), très inefficace !)

Pile avec tableaux :



Stack :

```
int[] T;
int top;
// créer empty(), full(), push(x), pop()

function empty() {
    return (top==0);
}
function full() {
    return (top==T.length);
}
function push(int x) {
    if (!full()) {
        T[top]=x;
        top++;
    }
}
function pop() {
    if (!empty) {
        int x=T[top-1];
        top--;
        return x;
    }
    return -1; // si la pile n'a que des
}              // valeurs positives.
```

Exercice 5. *Notation polonaise inverse.*
La *notation polonaise inverse* (RPN) ou *notation postfixe* est une notation pour les expressions arithmétique qui met un opérateur après ses opérandes et pas entre eux. Par exemple, l'expression $2 + 3$ s'écrit « `2 3 +` » en RPN, et l'expression $2 \times 3 + 4$ s'écrit « `2 3 * 4 +` ».

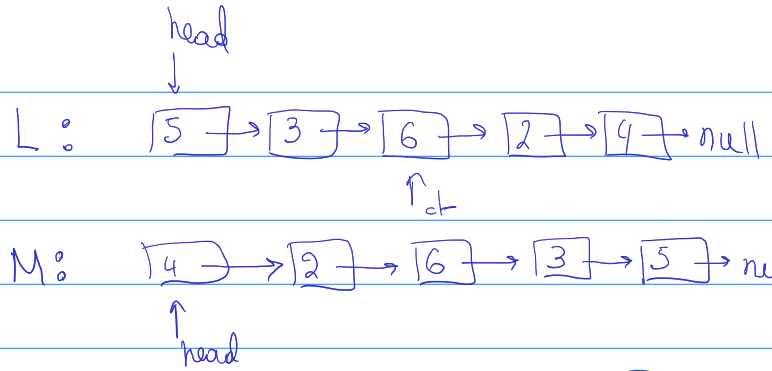
1. Écrivez les expressions suivantes en RPN :
   (a) $1 - 2 - 3$;
   (b) $1 - (2 - 3)$;
   (c) $1 \times 2 + 3 \times 4$;
   (d) $1 - 2 \times 3 + 4$.

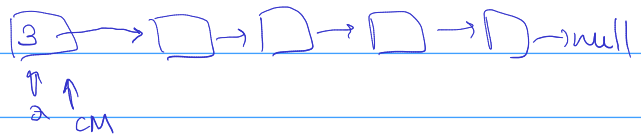
Une expression en RPN s'évalue très facilement à l'aide d'une seule pile [1] : un opérande correspond à un *push*, un opérateur correspond à deux *pop*, du calcul de l'opération, et d'un *push* du résultat.

2. Écrivez un algorithme qui, étant donnée une liste de nombres et d'opérandes représentant une expression en RPN, retourne la valeur de l'expression. Exécutez votre algorithme sur les expressions RPN que vous avez obtenues à la question précédente, et vérifiez qu'il produit le bon résultat.
3. Modifiez votre algorithme pour qu'il détecte les expressions incorrectes (il y a deux cas à gérer). Vous pouvez supposer donnée une fonction `error` qui termine l'exécution de l'algorithme.
4. Que faut-il changer pour ajouter un opérateur unaire de passage à l'opposé ? Peut-on utiliser le même symbole pour le passage à l'opposé et la soustraction ?
5. Écrivez une calculatrice RPN dans votre langage de programmation favori.

1. Il faut deux piles pour évaluer une expression en notation *infixe* habituelle.

Exo 4 :






```
fonction inverser (L) :
    Cellule cM = null
    Cellule cL = L.head;
    tant que (cL != null) {
        Cellule a = new Cellule(cL.data, null);
        a.next = cM;
        cM = a;
        cL = cL.next;
    }
    Liste M = new Liste(cM);
```

2.