afficher1sur2it( l: liste )
Donnée : l : liste à parcourir

début

tantque ( l != vide)
afficher ( l -> info)
l ← l -> succ
si l ≠ ϕ alors l ← l -> succ
Fin tq
fin

class Maillon { int info;
Maillon succ;
}

class liste. { Maillon premier}

_Affiche1sur2rec( l: m
Donnée : l : liste à p

deb

Si l = ϕ retourn
afficher ( l -> inf
l ← l -> succ
_Affiche1sur2rec ( l ←
fin

_Affiche_1sur2rec( l: list
_ afficher_1surrec( l -> pr

```
longueur_liste_circ ( l: liste ) : entier
donnée : l: liste circulaire à afficher
Var loc : taille: entier, tete: maillon
 Resultat: taille
debut
    Si l=ϕ alors taille←0
    Sinon
        tete ← l
        taille← 1
        tantque l->succ ≠ tete faire
            taille ++
            l ← l->succ
        Fin tq
    Fin si
    retourner taille
fin
```

Affiche_liste_inverse (l: liste)
    affiche_liste_inverse_rec (l.premier)

afficher_liste_inverse_rec(tete: maillon)
Donnée: tete: tete de la liste

Debut
    Si l=∅ retourner
    afficher_liste_invers_rec (tete → succ)
    afficher(tete → info)
Fin

1 concatener (l1 : liste, l2 : liste) : liste

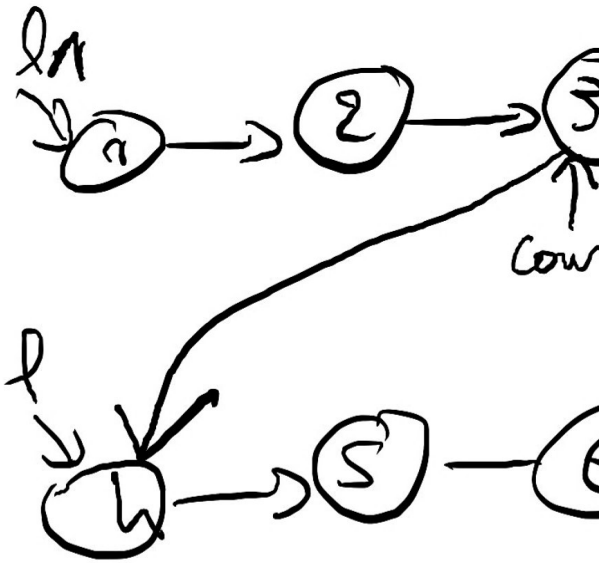
Donnees : l1, l2 : les listes à concatener

var loc : courant : maillon temporaire

Resultat : la liste concatenat° de l1 et l2

**debut**

```
Si l1 = ∅ alors retourner l2
Si l2 ≠ ∅ alors
    cour ← l1
    // avancer jusqu'au dernier maillon de l1
    tantque ( cour → succ ≠ ∅ )
        cour ← cour → succ
    fintq
    cour → succ ← l2
finS:
retourner l1
fin
```



2 Ajout ( l: liste, p: entier, e: entier )