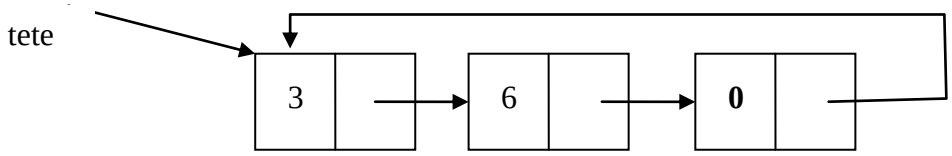
Liste chaînée circulaire - Fusion

Solution :

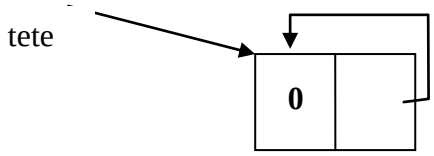
Une liste chaînée circulaire est une liste dont le dernier élément pointe sur le premier élément amenant ainsi la liste à pouvoir être parcourue en boucle.
Dans le cadre de notre exercice, nous allons lui adjoindre une sentinelle. Une sentinelle est un élément particulier, qui ne contient pas de valeur utile (nous lui donnerons la valeur 0), mais qui va symboliser la fin de notre parcours de liste.

Soit par exemple, une liste chaînée circulaire constituée d'entiers et triée dans l'ordre croissant avec sentinelle.
Elle contient, dans notre exemple qui suit, 2 valeurs (3 et 6) et la sentinelle (valeur à 0) qui pointe sur la tête de liste.



Question 1 *:*
Comment matérialise-t-on une liste vide avec ce procédé ? Faites un schéma.
Quel est donc le test qui permet de tester si une liste est vide ?
Quel est l'intérêt de la technique de la sentinelle ?

Une liste « vide » est une liste contenant un seul élément : la sentinelle et dont ce seul élément pointe sur lui-même.

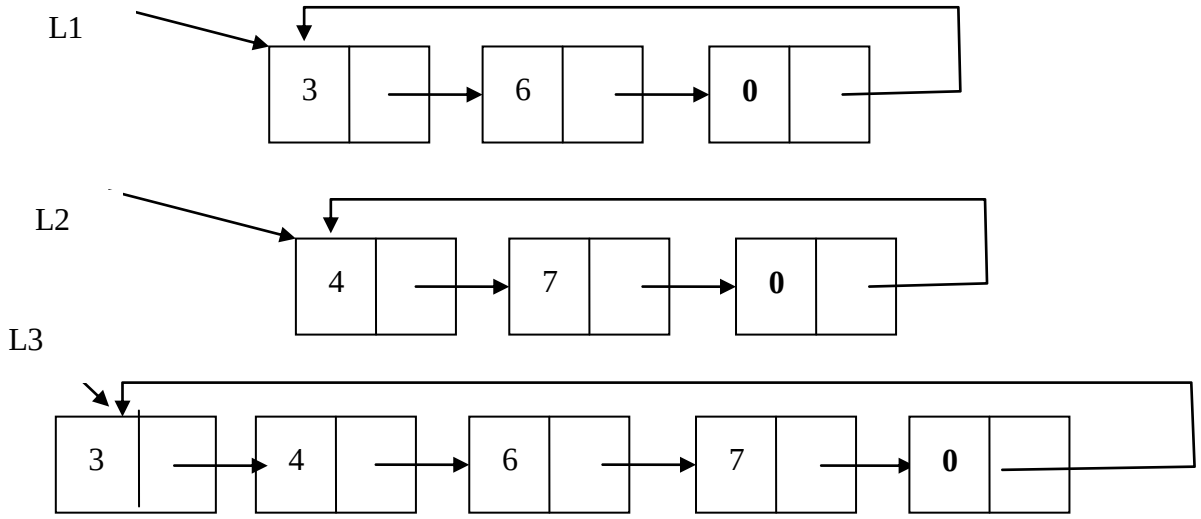


L'intérêt technique, lors de la gestion des listes (parcours, ajout, suppression), est de ne pas avoir à traiter du problème lié à la liste vide avant toute opération sur une liste. Les tests, nous le verrons lors de la suite de cet exercice, s'en trouvent simplifiés.
On évite notamment le risque de tester le contenu d'une cellule qui pointe sur null !!!

Question 2 :
Soient deux listes circulaires équipées de sentinelle.
Ecrire une procédure qui réalise la fusion de deux listes circulaires en une seule (elle-même triée).
Son entête est :

Procédure fusion_liste_circulaire ((E) L1 : ptelemnt, (E) L2 : ptelement, (S) L3 : ptelement)
// procédure qui fusionne deux listes en une seules. Les listes d'origine disparaissent.
// Les cellules sont récupérées et rangées directement dans la liste dont l'adresse de tête sera
// pointée par L3

Exemple :
L1 et L2 ont fusionné et donné L3



Type ptelement = ^ element
Type element = **Structure**
val : Entier
suivant : ptelement
finstructure

Procédure créer_sentinelle (E/S L : ptelement)
/ / Procédure qui crée une liste circulaire constituée d'une sentinelle*/
Début
L ← allouer (element)
L^.val ← 0
L^.suivant ← L
Fin procédure

Procédure insérer_élément (E/S L1, E/S L2 : ptelement)
/* procédure qui insère physiquement au début de L2 un élément positionné en tête de L1*/
déclaration
mem : ptelement
Début
mem ← L1^.suivant // on mémorise la position du suivant dans L1
L2^.suivant ← L1 // on met l'élément qui vient de L1 à la suite de L2
L2 ← L1 // le dernier élément de L2 est donc ce nouvel élément
L1 ← mem // L1 pointe désormais sur l'élément suivant de la liste
Fin Procédure

Procédure fusion_liste_circulaire (E L1 : ptelemnt, E L2 : ptelement, S L3 : ptelement)
// Procédure qui assure la fusion de 2 listes L1 et L2 dans une troisième pointée par L3
Déclaration MemDébut : ptelement
Début
Créer_sentinelle (L3)
MemDébut ← L3 // on mémorise l'adresse de la sentinelle pour la fin de la procédure

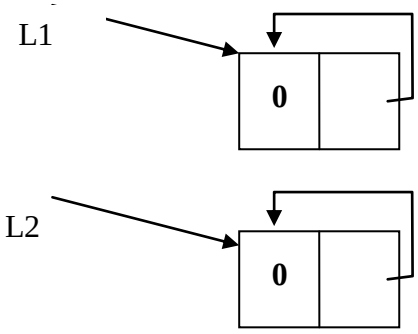
Tant que (L1^.val <> 0) et (L2^.val<>0) **faire** //aucune sentinelle en vue
Si (L1^.val < L2^.val) **alors**
Insérer_élément (L1,L3) // insérer l'élément en tête de L1 à la suite de L3
Sinon
Insérer_élément (L2,L3) // insérer l'élément en tête de L2 à la suite de L3
Finsi
Fin Tant que
Tant que (L1^.val <> 0) **faire** // La liste L1 n'a pas été vidée
Insérer_élément (L1,L3)
Fin tant que
Tant que (L2^.val <> 0) **faire**
Insérer_élément (L2,L3) // La liste L2 n'a pas été vidée
Fin tant que
Désallouer (L1) //rendre l'espace occupé par la sentinelle de L1
Désallouer (L2) //rendre l'espace occupé par la sentinelle de L2
L3^.suivant ← MemDébut // faire pointer dernier élément de L3 sur la sentinelle créée
L3 ← MemDébut^.suivant // la tête de liste L3 correspond au suivant qui avait été
// rangé dans la sentinelle après de sa création
Fin Procédure

<u>Question 3</u> :

Tester (numérotez les lignes de votre algorithme et indiquez le déroulement) cette procédure avec :
La procédure se numérote ainsi :

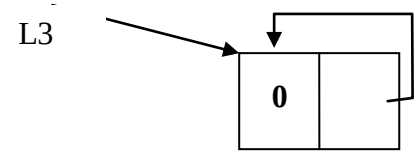
1. Procédure fusion_liste_circulaire (E L1 : ptelemnt, E L2 : ptelement, S L3 : ptelement)
2. Déclaration MemDébut : ptelement
3. Début
4. Créer_sentinelle (L3)
5. MemDébut ← L3
6. Tant que (L1^.val <> 0) et (L2^.val<>0) faire
7. Si (L1^.val < L2^.val) alors
8. Insérer_élément (L1,L3)
9. Sinon
10. Insérer_élément (L2,L3)
11. Finsi
12. Fin Tant que
13. Tant que (L1^.val <> 0) faire
14. Insérer_élément (L1,L3)
15. Fin tant que
16. Tant que (L2^.val <> 0) faire
17. Insérer_élément (L2,L3)
18. Fin tant que
19. Désallouer (L1)
20. Désallouer (L2)
21. L3^.suivant ← MemDébut
22. L3 ← MemDébut^.suivant
23. Fin Procédure

Déroulement sur le premier exemple



1. Procédure fusion_liste_circulaire (E L1 : ptelemnt, E L2 : ptelement, S L3 : ptelement)
2. Déclaration MemDébut : ptelement
3. Début
4. Créer_sentinelle (L3)

L'appel de la procédure produit la liste L3 suivante :



5. MemDébut ← L3

On mémorise l'adresse L3 dans MemDébut

6. Tant que (L1^.val <> 0) et (L2^.val<>0) faire

L1^.val et L2^.val sont toutes deux égales à 0 => on n'entre pas dans la boucle
On saute en 13

13. Tant que (L1^.val <> 0) faire

L1^.val égale à 0 => on n'entre pas dans la boucle, on saute en 16

16. Tant que (L2^.val <> 0) faire

L2^.val égale à 0 => on n'entre pas dans la boucle, on saute en 19

19. Désallouer (L1)
20. Désallouer (L2)

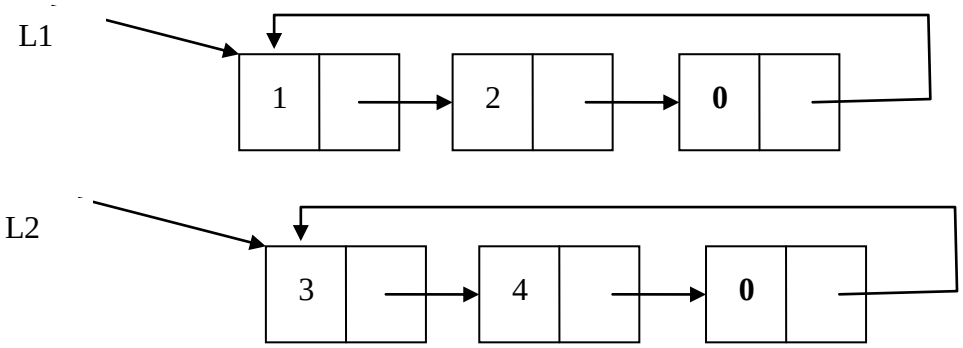
On a restitué l'espace occupé par les éléments pointés par L1 et L2

21. L3^.suivant ← MemDébut
22. L3 ← MemDébut^.suivant

L'élément de tête reste L3 qui pointe sur lui-même, ces affectations ne changent rien. Seule une sentinelle constitue la liste L3 (soit une liste vide fusion de deux listes vides).

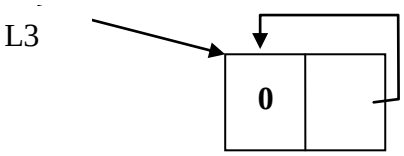
23. Fin Procédure

Déroulement sur le deuxième exemple



1. Procédure fusion_liste_circulaire (E L1 : ptelemnt, E L2 : ptelement, S L3 : ptelement)
2. Déclaration MemDébut : ptelement
3. Début
4. Créer_sentinelle (L3)

L'appel de la procédure produit la liste L3 suivante :



5. MemDébut ← L3

On mémorise l'adresse L3 dans MemDébut

6. Tant que (L1^.val <> 0) et (L2^.val<>0) faire

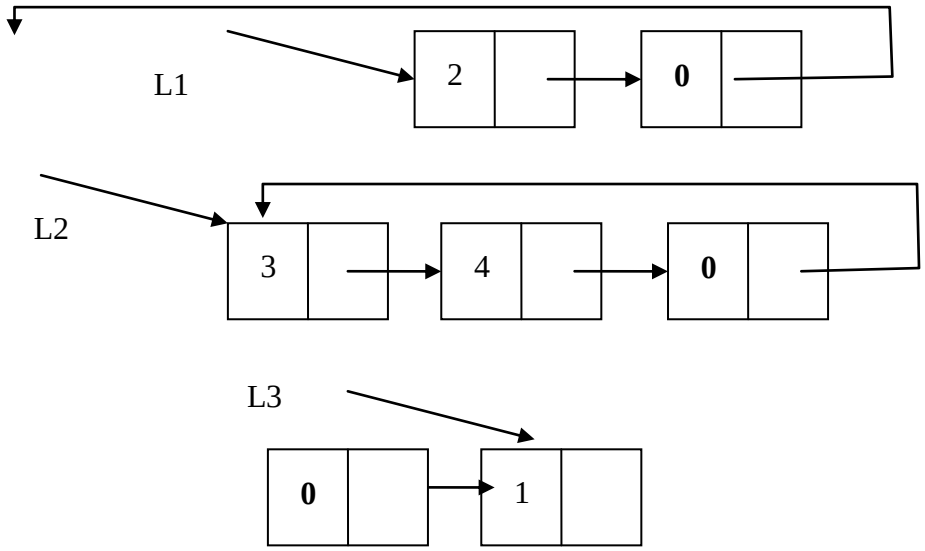
L1^.val et L2^.val sont différents de 0, on entre dans la boucle

7. Si (L1^.val < L2^.val) alors
8. Insérer_élément (L1,L3)
9. Sinon
10. Insérer_élément (L2,L3)
11. Finsi

L1^.val (1) < L2^.val (3) => on insère l'élément en tête de L1 à la suite de l'élément pointé par L3 grâce aux instructions (procédure appelée) suivantes :

mem ← L1^.suivant
L3^.suivant ← L1
L3 ← L1
L1 ← mem

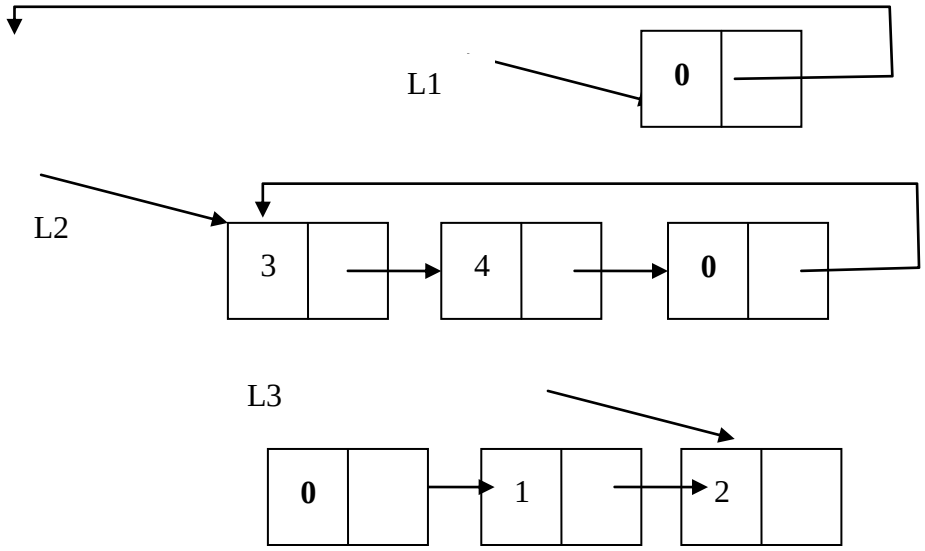
Les listes sont alors les suivantes :



L1^.val et L2^.val sont différents de 0, on continue dans la boucle

7. Si (L1^.val < L2^.val) alors
8. Insérer_élément (L1,L3)
9. Sinon
10. Insérer_élément (L2,L3)
11. Finsi

L1^.val (2) < L2^.val (3) => on insère l'élément en tête de L1 à la suite de l'élément pointé par L3 (procédure appelée):



7. Tant que (L1^.val <> 0) et (L2^.val<>0) faire

L1^.val = 0 => on sort de la boucle et on saute en 13

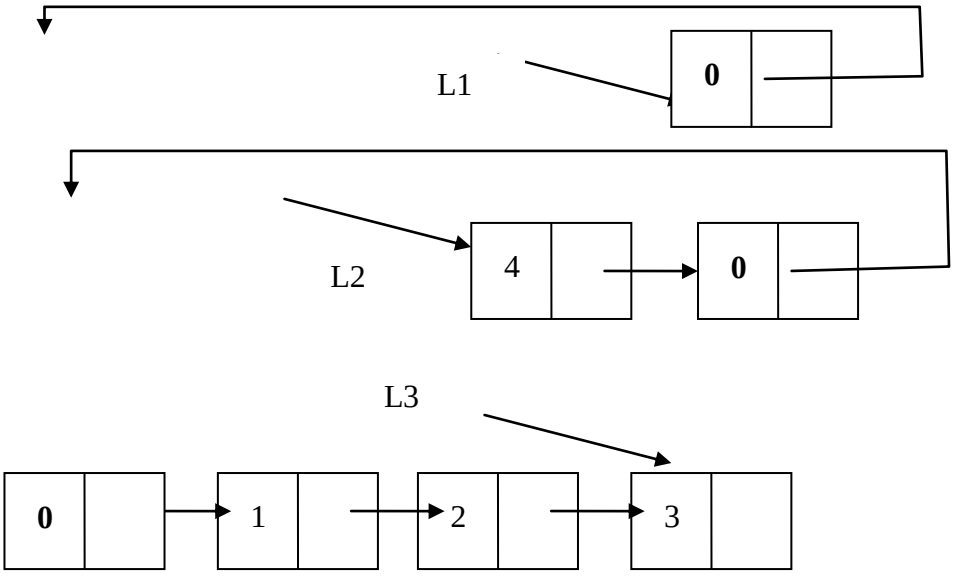
13. Tant que (L1^.val <> 0) faire

L1^.val égale à 0 => on n'entre pas dans la boucle, on saute en 16

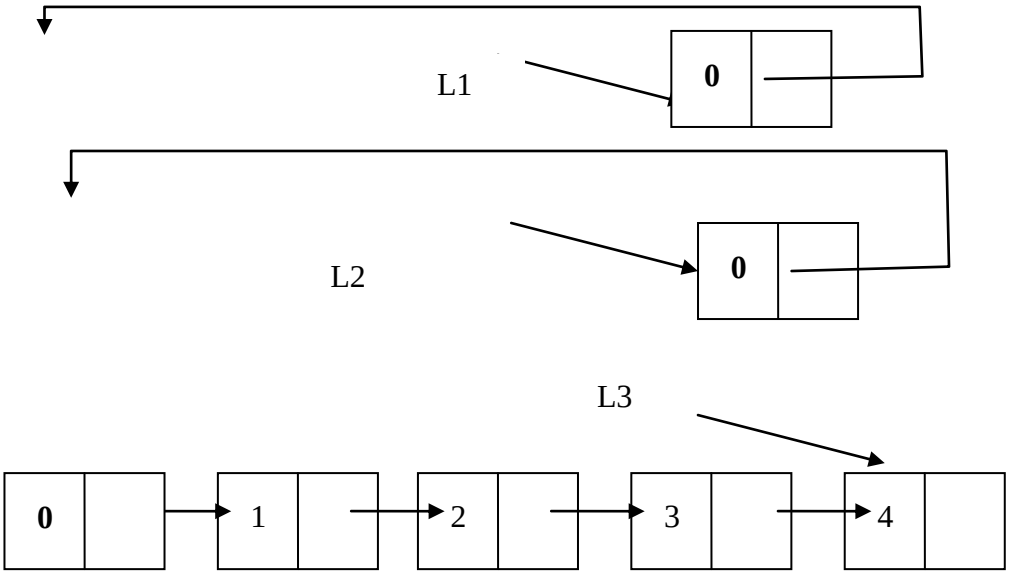
16. Tant que (L2^.val <> 0) faire

L2^.val différent de 0 => on entre dans la boucle

Insérer_élément (L2,L3)
mem ← L2^.suivant
L3^.suivant ← L2
L3 ← L2
L2 ← mem



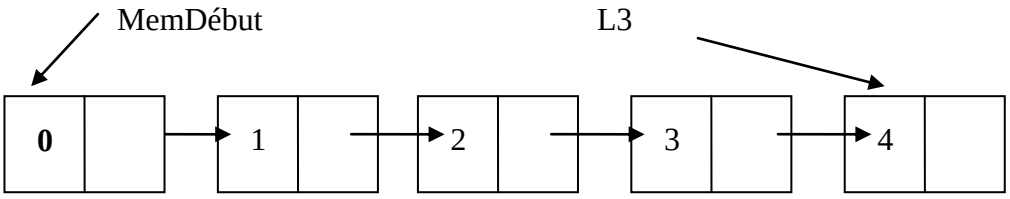
L2^.val différent de 0 => on continue la boucle. On obtient ainsi :



L2^.val égale à 0 => on sort de la boucle, on saute en 19

19. Désallouer (L1)
20. Désallouer (L2)

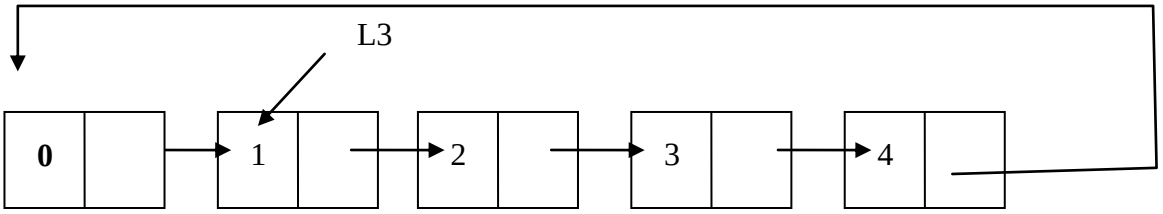
On a restitué l'espace occupé par les éléments pointés par L1 et L2. On a désormais



puis

21. L3^.suivant ← MemDébut
22. L3 ← MemDébut^.suivant

On a à la fin de cette procédure :



Soit une liste circulaire dont L3 point sur le premier élément et qui se finit par une sentinelle pointant elle-même sur le premier élément.