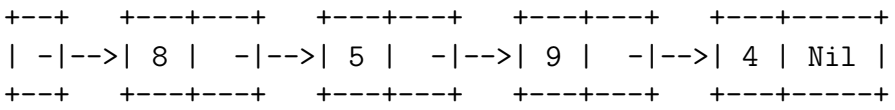
Listes chainées

1 Listes simplement chainées

Une liste chainée est une structure de données pouvant contenir plusieurs éléments. Chaque élément (noeud ou cellule) est structuré en deux champs : une partie *valeur* contenant l'information et une autre partie *adresse* contenant l'adresse du prochain élément.
L'adresse du premier élément de la liste chainée doit être sauvegardée dans un pointeur à part. Dans le cas d'une liste vide ce pointeur ce pointeur va contenir la valeur *Nil*.
Le dernier élément de la liste ne pointe sur rien (*Nil*).

Exemple Une liste chainée contenant les valeurs 8, 5, 9 et 4 dans cet ordre.



1.1 Déclaration d'une liste chainée simple

On commence par déclarer la structure élément :

```
type Element = strucuture
    champs_1: type_champs_1
    ...
    champs_1: type_champs_1

    suivant: ^Element
fin structure
```

Ensuite on déclare un nouveau type qu'on appellera *Liste* qui n'est rien d'autre qu'un pointeur sur une structure de type *Element*.

```
type Liste = ^Element
```

Déclaration d'une variable de type *Liste*.

```
var l: Liste
```

Exemple Déclaration d'un type liste chainée contenant des valeurs entières.

- En langage algorithmique

```
type Element = structure
    val: entier
    suivant: ^Element
fin structure

type Liste = ^Element
```

- En langage C

```
struct Element {
    int val;
    struct Element *suivant;
};

typedef struct Element* Liste;
```

Exemple Création d'une liste simplement chainée composée de deux éléments de type entier. Le premier élément de la liste va contenir la valeur 8 et le deuxième élément va contenir la valeur 5.

– En langage algorithmique

```
programme exemple
    type Element = structure
        val: entier
        suivant: ^Element
    fin structure

    type Liste = ^Element

    var l, p: Liste
debut
    allouer(l)
    l^.val <-- 8
    allouer(p)
    p^.val <-- 5
    p^.suivant <-- NIL
    l^.suivant <-- p
fin
```

– En langage C

```
#include<stdio.h>
#include<alloc.h>

typedef struct Element Element;

struct Element {
    int val;
    Element *suivant;
};

typedef Element* Liste;

void main() {
    Liste l, p;

    l = (Element*)malloc(sizeof(Element));
    l->val = 8;

    p = (Element*)malloc(sizeof(Element));
    p->val = 5;

    p->suivant = NULL;

    l->suivant = p;
}
```

1.2 Opérations sur les listes chainées

1.2.1 Opérations sans parcours de la liste

Soit l une liste chainée.

Liste vide Cette fonction a pour but de vérifier si une liste l est vide ou non. Elle retourne *vrai* si la liste est vide sinon elle retourne *faux*.

```
fonction vide(l: Liste): booléen
debut
    si l = NIL alors
        retourner vrai
    sinon
        retourner faux
    fin si
fin
```

Insérer un élement en tête de la liste Soit e une variable de type *Element* qu'on veut insérer à la tête de la liste l.

```
fonction inserer(l: Liste, var e: Element): Liste
debut
    e.suivant <-- l
    l <-- adresse de e
    retourner l
fin
```

Supprimer le premier élément de la liste

```
fonction supprimerTete(l: Liste): Liste
    var p: Liste
debut
    p <-- l
    l <-- l^.suivant
    désallouer(p)
    retourner l
fin
```

Fin de liste Une fonction qui nous renvoie une sous liste de l en enlevant le premier élément.

```
fonction fin(l: Liste): Liste
debut
    si l = NIL alors
        retourner NIL
    sinon
        retourner l^.suivant
    fin si
fin
```

1.2.2 Opérations avec parcours de la liste

Calcul de la longueur de la liste

– Version itérative

```
fonction longueur(l: Liste): entier
    var n: entier
debut
    si l = NIL alors
        retourner 0
    sinon
        n <-- 1

        tq l^.suivant <> NIL faire
            n <-- n + 1
            l = l^.suivant
        fin tq

        retourner n
    fin si
fin
```

– Version récursive (Exercice)

```
fonction longueur(l: Liste): entier
debut
    si l =  NIL alors
        retourner 0
    sinon
        retourner 1 + longueur(fin(l))
    fin si
fin
```

Insérer un élément à la fin de la liste

```
fonction insererFin(l: Liste, var e: Element): Liste
    var p: Lisste
debut
    e.suivant <-- NIL

    si l = NIL alors
        l <-- adresse de e
    sinon
        p <-- l
        tq p^.suivant <> NIL faire
            p <-- p^.suivant
        fin tq

        p^.suivant <-- adresse de e
    fin si

    retourner l
fin
```

1.3 Exercice

Écrire un algorithme qui permet de créer une liste simplement chainée d'entiers introduits par l'utiliser et puis afficher le contenu de cette liste.

1.4 Exercice

Soit l une liste simplement chainée, et e une variable de type *Element*. Écrire une fonction qui permet d'insérer e dans la liste l à la position k.

2 Listes chainées bidirectionnelles

Les listes simplement chainées peuvent être qualifiées de monodirectionnelles car on ne peut les parcourir que dans un seul sens de gauche à droite. Si on veut faire un parcours de droit à gauche il faut ajouter un pointeur permettant l'accès au noeud précédent. On qualifie alors la liste de bidirectionnelle.

Pour faciliter le parcours de droite à gauche, on mémorise l'adresse du dernier élément dans une variable qu'on appellera *fin*.

Ajouter la représentation graphique

2.1 Déclaration d'une liste bidirectionnelle

```
Type Element = Structure
    champs_1: type_champs_1
    ...
    champs_n: type_champs_n
    Suivant: ^Element
    Precedent: ^Element
Fin Structure

Type BListe = Structure
    Debut: ^Element
    Fin: ^Element
Fin Structure
```

2.2 Opérations sur les chaines bidirectionnelles

- Supprimer l'élément en tête de la liste
- Ajouter un élément en tête de la liste
- Insérer un élément en fin de liste

2.3 Exercice 01

Écrire une fonction qui permet de créer une liste bidirectionnelle d'entiers introduits par l'utilisateur, ensuite afficher le contenu de cette liste.

2.4 Exercice 02

Écrire une fonction qui permet d'insérer un élément e dans la liste l à la position k.

3 Listes circulaires

La liste circulaire est une sorte de liste simplement ou doublement chainée, qui comporte une caractéristique supplémentaire pour le déplacement dans la liste, *elle n'a pas de fin*. Pour rendre la liste sans fin, le pointeur suivant du dernier élément pointera sur le premier élément de la liste.

NB : Deux pointeurs, un sur le premier élément et le deuxième sur le dernier élément de la liste mais ce n'est pas une obligation.