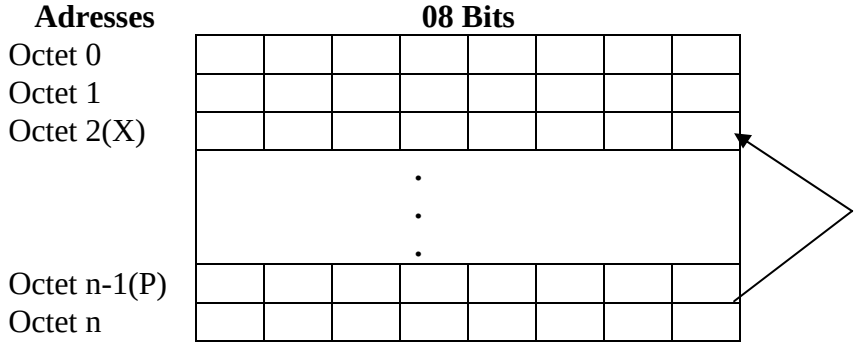
Chapitre 3:Les listes

1. Introduction

- La mémoire centrale est composée d'un très grand nombre d'octets.
- Chaque octet est repéré par un numéro appelé adresse de l'octet.



- Chaque variable dans la mémoire occupe des octets contigus, c'est-à-dire des octets qui se suivent.

Exemple :

float x ;

En C++ :

- Un float occupe 4 octets qui se suivent. L'adresse de la variable est l'adresse **de son premier octet**.
- On peut connaître l'adresse de la variable x par l'opérateur **&**.

float x ;
p = &x ; //adresse de la variable x : adresse de son premier octet
La variable **p** contient une valeur qui est **l'adresse** de la variable **x**

2. Allocation de la mémoire

- Les algorithmes (programmes) consomment essentiellement deux ressources :
  - ✓ Le temps d'exécution.
  - ✓ L'espace mémoire réservé.
- On distingue deux types d'allocation ou la réservation de l'espace mémoire:
  - ✓ Allocation statique.
  - ✓ Allocation dynamique.

2.1.Allocation statique :

- L'allocation de l'espace mémoire se fait **avant l'exécution** du programme, c'est-à-dire **après la compilation**.
- C'est donc, le cas de variable de type simples et de type tableau.

Exemple :
X : réel ;
A : entier ;
T[50] : tableau d'entier ;
Liste_Etudiants[1500] : Tableau d'Etudiant ;
X, A, T, Liste_Etudiants sont des ***variables statiques***

2.2.Allocation dynamique :

- L'allocation de l'espace se fait au fur et à mesure de l'exécution du programme.
- Pour pouvoir faire ce type d'allocation, l'utilisateur doit disposer des deux opérations : ***allocation*** et ***libération*** de l'espace.
- La majorité des langages de programmation offre cette possibilité.

Le système d'exploitation prévoir une partie de la MC à cet effet appelée le TAS (table allocation system).

Exemple :

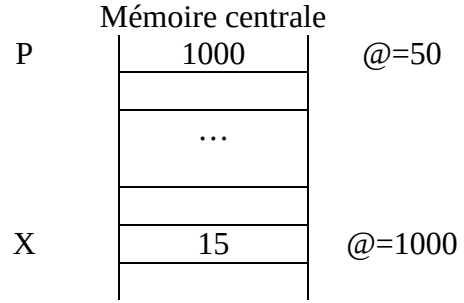
int X ;
P=&X ;
X est une variable dynamique

3. Le type pointeur

3.1. Définition :

- En programmation, Un **pointeur** est une variable contenant une adresse mémoire.

Exemple :



x= 15
P un pointeur contient l'adresse de x dans la mémoire
P= 1000

3.2.Déclaration d'un pointeur :

Symbole (*) pour le type pointeur

En algorithmique	En langage C++
*<Nom de la variable> : * Type de la variable ;*	*Type de la variable * <Nom de la variable> ;*
Exemples :	**Exemples :**
p : *****entier;	int * P ;
Q :*réel ;	float * Q ;

Exemple 1:

```
#include <iostream>
intmain ( )
{
int x=4 ;                //déclaration d'une variable entier x
int *p ;              //déclaration d'une variable pointeur p
p=&x ;                //p pointe x ( p contient l'adresse de x)
cout<<*p ;      // affiche le contenu de x
getchar();
return 0;
}
```

Résultat : 4

Example 2:

```
#include <iostream>
int main ( )
{
int x=4 ;              //déclaration d'une variable x
int *p ;               //déclaration d'un pointeur p
P=&x ;                 //p pointe x ( p contient l'adresse de x)
*p=15 ;              // la valeur 4 de x est remplacée par 15
cout<< x ;                          // imprime le contenu de x
Return 0 ;
 }
```

Résultat : 15

Exemple 3:

```
#include <iostream>
int main ( )
{
int x=4 ;              //déclaration d'une variable x
int *p ;               //déclaration d'un pointeur p
P=&x ;                 //p pointe x ( p contient l'adresse de x)
*p= -3 ;          // la valeur 4 de x est remplacée par -3
X = X*3 ;
cout<< X ;                         // imprime le contenu de x
Return 0 ;
 }
```

Résultat : -9

3.3.Opération sur les pointeurs

a) Allocation dynamique :

Syntaxe :

En algorithmique	En C++
P= Allouer (type)	P = **new** type;

- Allocation d'un espace de taille spécifiée par le type de P.
- L'adresse de cet espace est rendue dans la variable de type Pointeur P.

Exemples :

- Allocation d'une zone mémoire pour un entier :

```
int * p;
P = new int;
```

- Allocation d'une zone mémoire pour un tableau de 10 entiers :

```
int * tab;
tab = new int [10];
```

- Pour accéder aux trois cases de notre tableau on peut faire comme suit :

```
/* La première case */
(*tab)= 16;
/* La deuxième case */
*(tab + 1) = 12;
/* La troisième case */
*(tab + 2) = 11;
```

b) Libération d'un pointeur:

Syntaxe :

En algorithmique	En C++
Libérer (p)	delete (P);

- **libération** de l'espace mémoire pointé par P

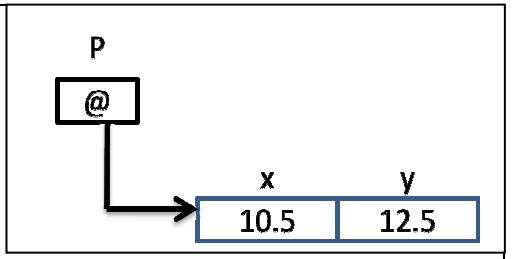
c) Pointeur et enregistrement :

- Pour Accéder a un champ d'un enregistrement pointé par un pointeur P, On utilise la notation « -> » :

Syntaxe : **P -> « champ »**

***Exemple*:**

```
Algorithme exp_PE
Type
Structure point
x:  réel;
y:  réel;
Finstructure
P: * point;
Début
P= Allouer (point); // allocation de l'espace mémoire
P-> x ← 10.8;
P -> y ← 12.5;
Ecrire (p -> x);       // affiche 10.8
Libérer(p);//libération de l'espace mémoire
Fin.
```



4. Structure de données « Liste »

4.1. Définition

- Une liste est une structure de données constituée d'une suite finie (éventuellement vide) ***d'éléments de même type***.
- Chaque élément de la liste est ***repéré*** selon leur ***rang*** dans la liste.

Exemple :
une liste d'entiers L = {11,-5,6,0}
le rang de l'élément **6** est **3**.

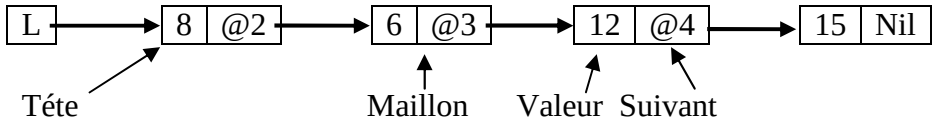
4.2. Représentation des listes

- Deux solutions possibles :

1) **Représentation contiguë :** on utilise des cellules contigües en plaçant des éléments dans un tableau.
2) **Représentation chainée :** elle consiste à utiliser des pointeurs pour relier les éléments.

4.3.listes chainées

- Une liste linéaire chaînée **Llc** est un ensemble de maillons (cases mémoire alloués dynamiquement) chaînés entre eux.
- Schématiquement, on peut la représenter comme suit :



- Un **Maillon** est une structure avec deux champs :
  - ✓ Champ **Valeur** contenant l'information.
  - ✓ Champ **Suivant** donnant l'adresse du prochain maillon.
- L'adresse de premier élément d'une LLC est souvent appelé *tête de la liste*.
- A chaque maillon est associée une adresse. Cette adresse est stockée dans le champ suivant du Maillon précèdent.
- Le champs suivant du dernier maillon pointe vers Nil (constitue l'adresse qui ne pointe aucun maillon).

4.4.Déclaration :

- En langage algorithmique :

```
Type
Structure Nom_Type_Maillon
Elément: Typeqq ;
Suivant : * Nom_Type_Maillon;
FinStructure ;
Type
Liste :*Nom_Type_Maillon;// le type liste qui désigne tout
                                    Pointeur vers un maillon
L : Liste;// équivalent à  L: * Maillon;
L, P, Q : Liste ;
```

Typeqq : désigne un type quelconque(int, float, personne, étudiant, Produit ...etc).

Les déclarations :
L, P, Q : Liste ; // signifie que L, P, Q sont des pointeurs vers des maillons

- En langage C ++:

```
TypeDef
struct Nom_Type_Maillon
{
Typeqq Elément;
Nom_Type_Maillon  *  Suivant;
};
typedef
Nom_Type_Maillon*Liste;
Liste  L, Q, tete;  //  équivalent à  Nom_Type_Maillon * L, Q, tete;
```

***Exemple* 1:** liste chainée d'entiers

```
Type
Structure Maillon
Ele : entier;
suivant: * Maillon;
finstructure
Type Liste : * Maillon;
LE : Liste ;
```

***Exemple* 2:** liste chainée d'étudiant

```
Type
Structure étudiant
 …
fin structure
Type
Structure Maillon
Ele : étudiant;
suivant: * Maillon;
fin structure
TypeListe : * Maillon;
Ou :
L :liste ;
```

***Exemple* 3:** liste chainée de personnes

```
Type
Structure Personne
nom: chaine de caractères
prénom: chaine de caractères
âge : entier
fin structure
Type
Structure Maillion
Ele : personne;
suivant: * Maillion;
fin structure
Type Liste : * Maillon;
```

***Remarque* :**

- L'accès à un champ d'une structure se fait par le biais du point pour les variables ordinaires et par le biais du -> pour les variables de type pointeur.

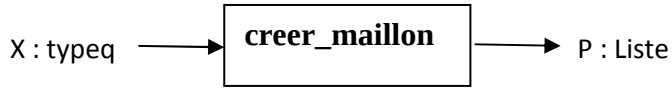
***Exemple* :**

X : Etudiant ; alors X.nom

Y : *Etudiant ;alors Y->nom.

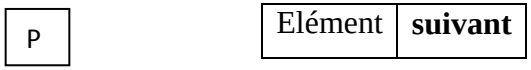
4.5.Opérations sur les listes :

a) **cree_maillon**: crée un nouveau maillon contient comme valeur x et retourne un pointeur contenant son adresse.

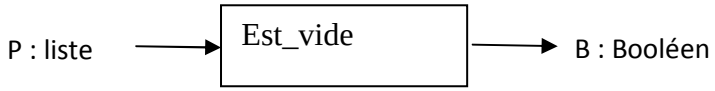


Rôle : créer un nouveau maillon

```
Fonctioncreer_maillon (x : typeq): Liste
    P: Liste   //  ou P: *Maillion
Début
   P ← Allouer (Maillon) ;   ……(1)
   P - > Élément ← x;……(2)
   P -> suivant   ← Nil;   ……(3)
   Retourne (P);
Fi n
```



b) **Est_vide :** teste si la liste est vide ou non



Rôle : teste si la liste est vide ou non

```
Fonction Est_vide(L : Liste) : Booléen
Début
Si (L=Nil) alors
   Retourner (vrai);
Sinon
   Retourner (faux);
Finsi
Fin.
```

c) **Premier** : retourne le premier élément de la liste L

Rôle : retourne le premier élément de la liste L

```
Fonction Premier (L: liste): typeq ;
Début
Si (L== Nil) alors
Ecrire (« la liste est vide ») ;
Sinon
Retourne (L ->ele);
Finsi
Fin
```

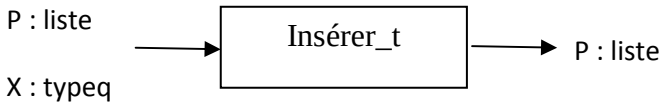
d) **Insertion d'un élément :** l'insertion d'un nouvel élément dans la liste Llc consiste en :

1) Création d'abord du maillon correspondant,
2) Affectation de la valeur a l'élément,
3) Puis le chaînage du nouvel élément avec la liste Llcsoit :
   - ✓ ***En tête de la liste* :** Pour le rajout du nouveau maillon au début de la liste on doit changer à chaque fois l'adresse de la tête.
   - ✓ ***En fin de liste* :** Pour le rajout du nouveau maillon à la fin de la liste on doit garder toujours deux pointeurs : la tête de la liste et la queue (adresse du dernier élément) de la liste.
   - ✓ ***Au milieu de la liste* :** Pour le rajout du nouveau maillon au milieu de la liste on doit d'abord

-Rechercher sa position (l'adresse du maillon qui va le précéder p et celle du maillon qui va le suivre)

- Puis tranché le chaînage entre p et s.
- Désormais p point vers le nouveau maillon et le nouveau maillon pointe vers s.

Insertion d'un élément en tête de la liste :



Rôle : Insérer un élément en tête de la liste

```
Fonction Insérer (x : typeq, L: Liste): Liste.
 P: Liste ;  //  ou P: *Maillion ;
Début
P ←creer_maillon (x);
P -> suivant   ← L;
 Retourne (P);
Fin ;
```

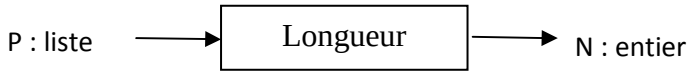
```
Procédure Insérer (x : Élément, var L: Liste)
 P: Liste ;
Début
   P ←creer_maillon (x);
   P -> suivant   ← L;
   L← P;
Fin ;
```

e) **reste (L: Liste):** retourne la liste L sans le premier élément. Elle retourne l'adresse de l'élément (maillon) suivant de la liste.

Rôle : retourne la liste L sans le premier élément

```
Fonction Reste (L: Liste): Liste
Début
Retourne (L -> suivant);
Fin
```

f) **Longueur de la liste: retourne** le nombre des éléments de la liste L.



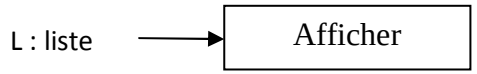
Rôle : retourne le nombre des éléments de la liste L

```
Fonction Longueur (L: Liste): entier        // version récursive.
Début
Si (L=Nil) alors
Retourne ( 0);
Sinon
  Retourne (1 + longueur (Reste (L)));
Finsi
Fi n
```

```
Fonction  Longueur (L: Liste): entier        // version itérative.
      Courant : Liste ;   // courant : *maillon ;
      Nb: entier ;
Début
       Nb←0;    courant←L;
      Tantque courant != Nil faire
          Nb ← Nb+1;
```

```
        courant←courant ->suivant ;
    Fintantque
    Retourner (Nb);
Fi n.
```

g) Affichage : afficher les éléments d'une liste.



Rôle : afficher les éléments d'une liste.

```
Procédure afficher (L: Liste)          // version récursive.
Début
   Si (L!=Nil) alors
        Écrire (premier (L)) ;
    Sinon
       Écrire  (premier(L)) ;
       Afficher (reste (L)) ;
  Finsi
Fin ;
```

```
Procédure afficher (L: Liste): entier        // version itérative.
    Courant : Liste ;   // courant : *maillon
    Nb: entier ;
Début
   Tantque (courant != Nil) faire
        Écrire (courant -> Ele);
       Courant ← courant ->suivant ;
  Fintantque
Fin ;
```

h) **Suppression d'un élément :** La suppression consiste d'abord en la rupture du chaînage du maillon concerné depuis la liste, selon l'un des trois cas, puis la libération de ce maillon :

- ✓ ***Supprimer le premier élément de la liste :*** Changement de la tête de la liste pour qu'elle pointe l'élément suivant de la tête actuelle.
- ✓ ***Supprimer un élément au milieu de la liste :*** Soit X l'élément à supprimer et soit p l'élément qui le précède et v l'élément qui le suit alors dorénavant v sera le suivant de p.
- ✓ ***Supprimer le dernier élément de la liste :*** l'avant dernier éléments de la queue pointera Null.

```
Procédure supprimer (Var L : Liste)
   P : Liste ;
Début
   P ←L ;
   L ← L->suivant ;
   Libérer (P) ;
Fin.
```

NB : Ce cas peut être intégré dans le cas N=°2 en supposant que le suivant de la queue est l'élément Null.

2.6. Types de listes chainées

a) Simple

b) Symétrique ou doublement chainée

c) Circulaire simple