L.L.C : un moyen dynamique

» Allocation Dynamique & Statique :

- Se fait au fur et à mesure de l'éxecution du programme.
  - allocation
  - libération de l'espace.
- la taille de mémoire à Allouer puisse être déterminée au moment de la compilation

» allocation de variables :

Création d'une VAR → réservation d'un espace mémoire.

- Statique : automatiquement par le Système
- Dynamique : manuellement par le programmeur

Modification par indirection ⇕ Utilisation des pointeurs

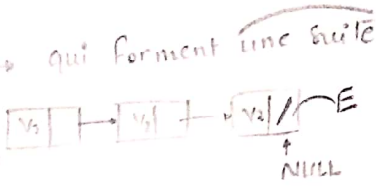
⚠ : retourne une adresse.

malloc (m) : alloue une zone m de taille m

free (p) : détruit la variable pointée par p

» Définition d'une L.L.C :

- Structure de donnée qui représente un ensemble des valeurs
- une séquence ordonnée qui forment une suite.



■ L.L.C se caractérise par :

- premier élément de la liste - tête de la liste -
- la fin de la liste

> utilise uniquement ses besoin.
> peuvent être détruite à tout moment
> Simplification.

■ Le type d'un maillon :

```
Struct Maillon ;
{ int val ; → Sa valeur
← Struct Maillon * Suiv ; → l'adresse du maillon suivant
};
```

Pointeur d'un maillon (maillon suivant.

> rappelons :
l'ADR d'une VAR est le numéro de sa première case. TAB

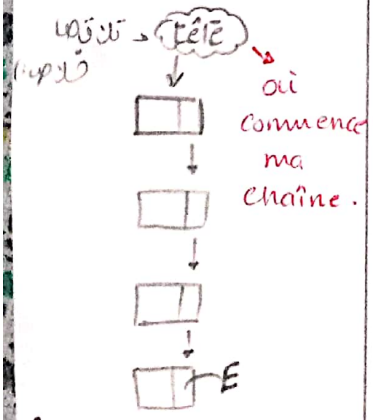
> p = &x
on dit p point sur x
*p = x ⚠

> NULL, 0 : p ne pointe aucune variable.

⚠ : une VAR Dynamique n'a pas du nom, et on peut la manipuler qu'à travers son ADR (Pointeur fin)

> élément d'une L.L.C → maillon
- Sa Valeur
- ADR du maillon suivant

le plus souvent dynamique



la tête :
> l'adresse de première cellule qui détermine la liste.

⚠ : free(p) ...

hey :

pour allouer une zone mémoire de 100 entiers avec malloc.

p = malloc (100*sizeof(int))

un tableau est un pointeur !

NULL = NIL

c'est une machine virtuelle

MLLC

(*P).Val ← V

(*P).Suivant ← V

on vas construire une procédure.

ex important :

```
int *p;
int i;
p = malloc (100 * sizeof(int)) :3;
// allouer une zone de taille 100 entiers
// on manipule cette zone comme si un tableau
```

p[0] = 10 ; p[1] = 3 ; p[2] = 5 ; ou alors ⟺ *p = 10 ; *(p+1) = 3 ; *(p+2) = 5

>> les modèles :

Allouer (p) : alloue (réserve) un nouveau maillon et affecte son adresse dans p.

Libérer (p) : détruit et libère l'espace occuppé par le maillon pointé par p

Aff_val (p, v) : affecte v dans le champ val du maillon pointé par p.

Aff_adr (p, q) : affecte q dans le champ adr du maillon pointé par p.

Valeur (p) : fonction qui retourne le contenu du champs val m.pointé par p.

Suivant (p) : fonction qui retourne le contenu du champ adr du m. pointé par p.

Des exemples :

> insertion d'une valeur v dans une liste L ordonné en ordre croissant :

```
ins_ord (v : typeqlq, var L : ptr)   // la nouvelle tête
                                     // on a dit qu'une liste est définie par sa tête pointeur
p, q, n : ptr
trouv : boolean.
```

Pour que notre liste reste ordonné.

```
Début : // on commence à rechercher l'endroit où elle doit être
    trouv ← FAUX.
Parcours ← p ← L   /* reçoit la tête, le début de la liste */
    q ← NIL.
```

q le précédent de p.

lieu :

ex important :

TQ (P <> NIL ∧ Non trouv) → la liste n'est pas encore finie ∧ on a pas trouver un element "Valeur" Valeur) = V

D

n'oublions pas que notre liste est ordonnée. ←

Si (V < Valeur (P))
    trouv ← vrai

Sinon

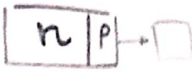
on passe au maillon suivant ←

    q ← p // on sauvgarde dans q l'adr du maillon
    p ← suivant(p) // avant du passer au prochain avec p.

FSI

on vas insérer V entre les maillons pointés par p & q

***/ insersion :**



بنبتي برك :: متوصل بيه

Allouer (n) // nouveau maillon.

aff-val (n, v)

aff-adr (n, p) // le suivant de n est p.

on a construit notre chaine! ←

Si (q <> Nil) // si il existe un maillon qui précède p.

    aff-adr (q, n) // le suivant de q devient n

Sinon.

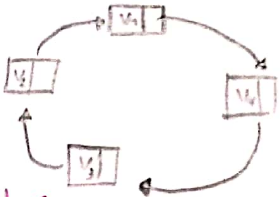
    L ← n // n devient la tête

>> Liste particulières :

> Liste circulaire :

①: La tête de la liste est l'adr du n'importe quel maillon de la liste circulaire

une liste où le dernier élement de la liste pointe sur le premier.



> Liste bidirectionnelles :

une liste qu'on peut la parcourir dans les deux sens.

La structure d'un maillon est : ←

{val : typqlq, adrg, adrd : pointeur

Sauv du tête & Queue pour simplifier le parcours dans les deux sens.

on ajoute un modele :

précedent (p).