

**République Algérienne Démocratique et Populaire**  
**Université des Sciences et de la Technologie Houari Boumediene**

**Faculté d'Electronique et d'Informatique**

**Département Informatique**



# **Les Pointeurs**

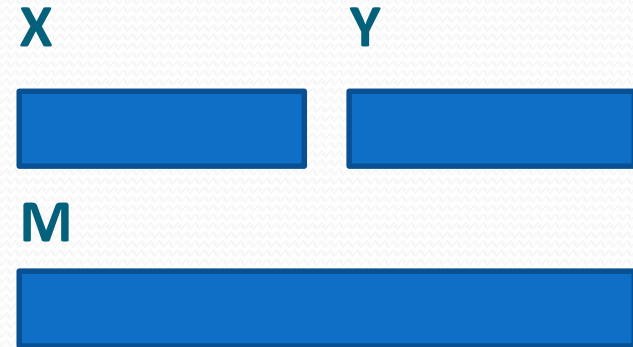
**Cours Algorithmique de 1ere Année MI**

**Présenté par : Dr. B. BESSAA**

# Variables et Adresses

Soit l'algorithme suivant:

```
Algorithme Exemple;  
Var      X,Y:entier;  
         M:reel;  
  
Debut  
    Ecrire("Donner deux entier : ");  
    Lire(X,Y);  
     $M \leftarrow (X+Y)/2$ ;  
    Ecrire("Moyenne =",M);  
  
Fin.
```



Dans la partie déclaration de cet algorithme, on a déclaré deux variables entières **X** et **Y** et une variable réelle **M**.

Donc au début de l'exécution:

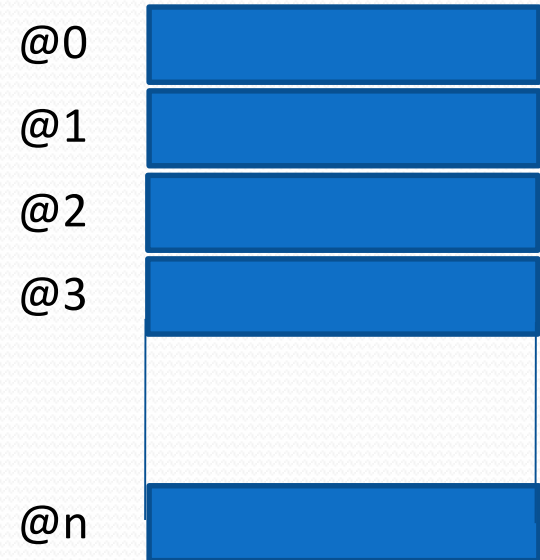
on va réserver deux emplacements mémoires nommés **X** et **Y** qui vont recevoir des entiers :donc de même taille.

Et un emplacements mémoire nommé **M** qui va recevoir un réel.

## Qu'est ce qui se passe en mémoire ?

La mémoire est un espace de stockage organisé en emplacements de même taille (appelés **Mots**). Chaque mot est identifié par une **Adresse**.

**Et justement**, le processeur, lorsqu'il exécute un programme (algorithme), communique avec la mémoire à travers les **adresses** et non pas les **noms** des variables déclarées dans le programme.



Donc, au fait, lorsque on réserve une variable **X**, **Y** ou **M**, on ne fait que **correspondre** ces noms de variables aux **adresses** mémoires des emplacements réservés.

# Qu'est ce qui se passe en mémoire ?

## Mais les variables déclarées ne sont pas toutes de même taille ?

**Oui**, effectivement, chaque type possède une **Taille** spécifique qui est un **multiple** de la taille d'un emplacement mémoire (taille élémentaires qui peut être **1 Octet** par exemple). Donc une variable peut être représentée sur plusieurs emplacements.

**Et alors, dans ce cas à quelle adresse va correspondre la variable ????**

**C'est simple**, la variable correspond à **l'adresse** du **premier** emplacement de l'ensemble des emplacements affectés à cette variable

Dans notre exemple, supposons qu'un entier occupe **2** emplacements et qu'un réel occupe **4** emplacements.

Si on suppose que les emplacements libres commencent à partir de l'adresse **@2**, on aura:



**X** correspond à **@2**      **Y** correspond à **@4**      **M** correspond à **@6**

# Qu'est ce qui se passe en mémoire ?

Et quand on déclare un Tableau, est-ce que chaque élément aura une adresse correspondante?

Bonne question.

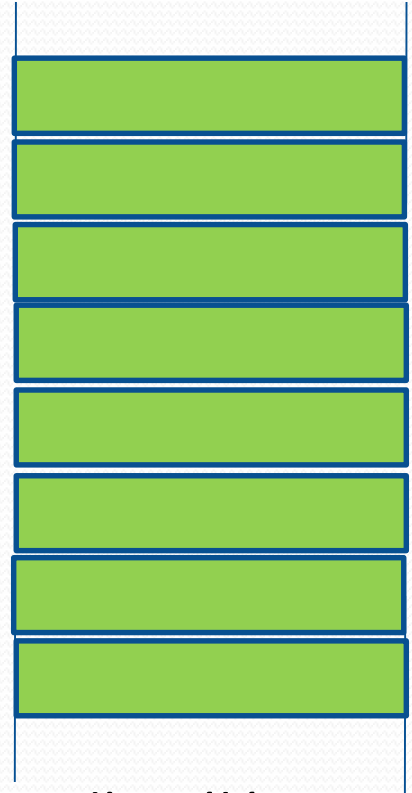
Alors, pour les tableaux, on fait correspondre une adresse au **nom** du tableau, c'est ce qu'on appelle l'adresse de **base**. Ensuite l'adresse de chaque élément du tableau est **calculée** suivant son **indice**, sachant que l'adresse du premier élément est égale à l'adresse du tableau.

T[1] : @1

T[2]:@1+2

T[3]:@1+4

T[4]:@1+6



Soit la déclaration **T:Tableau[4] de entier;**

Si on fait correspondre l'adresse **@1** au tableau, alors l'adresse d'un élément **T[I]** est calculée comme suit:

**@ = N\*(I-1)+@1**, où **N** est le nombre d'emplacements réservés pour un élément du tableau (exemple pour les entiers N = 2)

**Remarque:** Si les indices commencent à 0, alors : **@ = N\*I+@1**

Qu'est ce qui se passe en mémoire ?

C'est bon, mais comment ça se passe pour les enregistrements?

Presque la même chose.

Sauf que dans un enregistrement, l'adresse d'un champs dépend de l'adresse de **base** est **des champs précédents**.

Soit:

Etud:Enregistrement

Mat : entier;  
Nom:chaîne[20];  
Moy:reel;  
Sec:caractère;

Fin;

Si on fait correspondre l'adresse **@1** à Etud, alors l'adresse de **Mat** est @1 (@ du premier champs=@ de Etud (base))

**Nom** est @2=@1+taille(Mat) = @1 + 2

**Moy** est @3= @1+Taille(Mat)+Taille(Nom)=@1+2+20 = @1+22

**Sec** est @4= @1+Taille(Mat)+Taille(Nom)+Taille(Moy)=@1+26

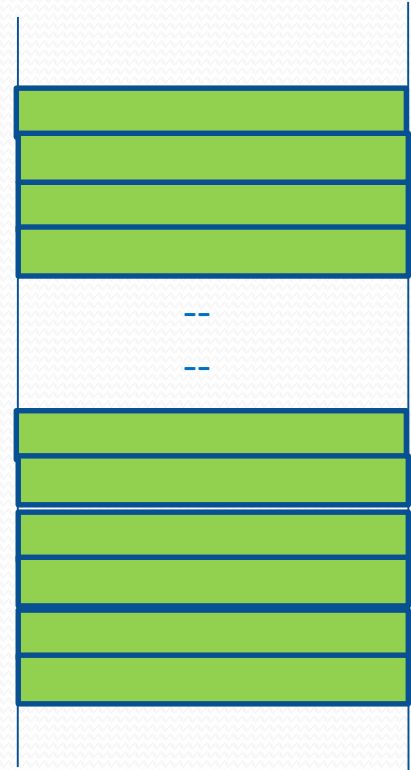
Etud ,Mat: @1

Nom :@1+2

@1+21

Moy :@1+22

Sec :@1+26



**Attendez attendez ! Entier, Réel, Tableau,... : pour tous ces types on fait correspondre une seule Adresse ???**

**Très bonne remarque.** Exactement, indépendamment du type, à toute variable on fait correspondre **une seule adresse** : c'est l'adresse du début de l'emplacement réservé.

**Et les adresses, ont-elles toutes la même taille???**

**Oui**, une adresse c'est comme un N° de téléphone, une carte de visite, ils ont la même taille, mais un **N° de téléphone** par exemple peut correspondre à une personne, une administration, une usine, un hôpital... qui ont des tailles différentes.

0888 546220



0888 546221



0888 546222





**Puisque à chaque variable correspond une adresse, peut-on utiliser l'adresse au lieu du nom ?**

**Oui, pourquoi pas.** Si vous chercher une personne par exemple, ou bien vous avez son **nom** et vous allez chercher son domicile, ou alors vous avez son **adresse** et vous allez tout droit au but.

**Non non, je veux dire en informatique: dans un programme ou dans un algorithme ?**

Alors toute cette histoire pour dire non, **bien sûre que oui.**

**Mais comment faire la différence entre une variable et son adresse ?**

**Ok,** donc on doit avoir une différence. Et bien on va ajouter un petit symbole pour dire que c'est une adresse, par exemple **@**

**Exemple**

**X:entier;**

alors **@X** est l'adresse correspondante à **X**

**M:reel;**

alors **@M** est l'adresse correspondante à **M**



**Est-ce que la valeur de X est égale à la valeur de @X ?**

**NON**, la valeur de **X** c'est la **donnée** stockée en mémoire, par contre la valeur de **@X** est l'**dresse** de l'emplacement mémoire.

### Exemple

**X** est la variable

Valeur de **X** est : -150

Valeur de **@X** est l'adresse: 765125

**Peut-on sauvegarder l'adresse de X?**

**Heureusement**, il suffit de mettre par exemple : **P ← @X**;

**Mais c'est quoi ce P ?**

**Et bien c'est une variable**, vous avez dit **sauvegarder**, donc il faut avoir un **emplacement** mémoire où le mettre.

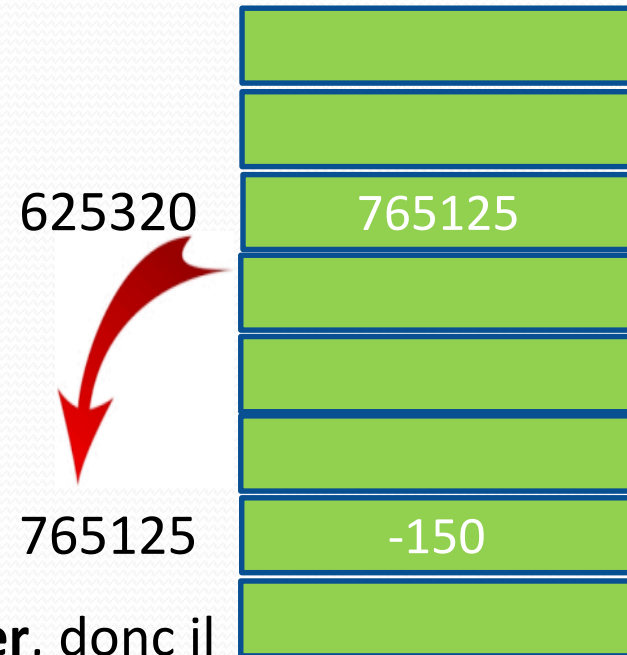
### Exemple

Valeur de P est : 765125

Valeur de @P est : 625320

**Et dans ce cas P nous oriente vers @X**

On appelle **P** : **pointeur** et **X** : **pointé**



Mais dans ce cas il faut déclarer ce « P », alors c'est quoi son type?  
Entier, Reel, ...

Effectivement, chaque variable utilisée doit être déclarée. Alors dans ce cas son type n'est ni Entier, ni Reel, ni ... C'est justement le type

# Pointeur

Pointeur est donc un **type** de variable spécifique qui ne peut contenir que des **adresses**.

Et alors comment on le déclare ?

Comme toute variable, un pointeur doit avoir un **Nom**, un **Type** et une **Valeur**.  
Sa **valeur** est toujours une adresse mémoire.

Par contre son **type** dépend de la variable sur laquelle il va **pointer**.

Pour dire que c'est un pointeur on utilise le symbole « ^ ».

## Syntaxe

<IdPtr> : ^<TypeVar>;

## Exemple

**Pent : ^entier; Pr: ^reel;**

Et si on a un type Tetudiant=**Enregistrement**

Matricule:entier;

Nom,Prenom:chaine[20];

**Fin;**

On peut déclarer:

**Petud: ^Tetudiant;**

Et si on déclare

X:entier; Y:reel;

Etd:Tetudiant;

On peut faire :

Pent ← @X; Pr ← @Y;

Petud ← @Etd;

Pour accéder aux valeurs, on écrit : **Pent^; Pr^; Petud^**

Après les affectations précédentes:

**Ecrire(X);** est équivalent à **Ecrire(Pent^);**

**Ecrire(Y);** est équivalent à **Ecrire(Pr^);**

**Ecrire(Etd.Nom);** est équivalent à **Ecrire(Petud^.Nom);**

**Oui, mais on était bien sans ces POINTEURS ? Ils apportent quoi de plus ? On peut bien manipuler nos variables sans les pointeurs ! C'est une double déclaration !!!**

Ok, on va voir.

Jusqu'à maintenant, toutes la mémoire que nous avons utilisée dans nos programmes était réservée avant l'exécution à l'aide des déclarations de variables.

Donc avant de commencer les traitements, on connaît exactement les emplacements mémoires (**adresses**) des variables utilisées, elles sont là intouchables. C'est ce qu'on appelle des variables **STATIQUES**

Pourquoi **STATIQUES** ???

Oui, car elles habitent le même endroit durant toute leur vie, elles n'ont pas **changé d'adresse**.

**Bon, mais on n'a pas le choix ! Vous nous avez toujours dit qu'il faut déclarer toutes les variables utilisées dans l'algorithme.  
Mais quelle est la relation entre ça et les pointeurs ?**

Patiencez, vous allez voir l'intérêt de ces **POINTEURS**.

Supposons que vous êtes un groupe de **10** étudiants, et vous voulez organiser un voyage ensemble. Donc une semaine avant le départ, vous allez faire une **réservation** d'hôtel pour **10** personnes que vous **payez à l'avance**.

Le jour **J**, **4** étudiants se sont excusés, ils ont des empêchements majeurs. Les **6** restants décident de partir.

**Mais malheureusement**, la direction de l'hôtel ne va pas **rembourser** les frais de réservation des **4**.

C'est le problème qui se pose pour toute réservation à l'avance (**Statique**: l'adresse de l'hôtel est connue **avant le départ**).

En **algorithmique**, on a vu ce type de problème dans la déclaration des tableaux. On déclare un tableau de taille **Max**, mais à l'exécution, suivant la taille exacte du tableau (le fameux **N**), il se peut que la moitié du tableau ne sera pas utilisé, mais l'espace est réservé durant toute l'exécution, et donc la moitié de l'espace est **réservé pour rien** !!!!

**Et alors qu'apporte les pointeurs pour résoudre ce problème?**

Bien, revenant au problème du voyage.  
On propose une **deuxième** solution.

Au lieu de faire une réservation à l'avance, le groupe décide de chercher un hôtel une fois arriver à destination.

Dans ce cas, le jour J, les **6** étudiants vont chercher un hébergement pour **6** seulement (**nombre exacte**) et non pas pour **10**. Et donc il vont payer pour **6** aussi et non pas pour **10**.

**Oui, mais dans ce cas on risque de ne pas trouver 6 places dans le même hôtel !!!**

**Effectivement**, il se peut qu'ils vont se répartir sur deux, trois ou à la limite **6** hôtels.

**Par exemple:**

**1ere** Hôtel : 2 Places, **2eme** Hôtel : 3 Places, **3eme** Hôtel : 1 Place

Donc ils seront tous hébergés, mais à des endroits (**Adresses**) différents. Et ces adresses ne sont connues qu'à **l'arrivée** à la destination (**A l'exécution**). Et ainsi on gagne les frais de réservation de **4** personnes.

C'est ce qu'on appelle une réservation **DYNAMIQUE** : l'adresse n'est pas connue à l'avance mais à l'exécution.

**Ok, tout ça est bon, mais on ne voit toujours pas la relation avec les POINTEURS.**

C'est parce qu'on n'a pas tout vu sur les pointeurs.

Et bien, lorsqu'on déclare un pointeur dans un algorithme, on **ne fait pas** de **réserve** d'emplacements mémoire, c'est juste une **prévision**: on déclare qu'on **aura besoin** d'un espace pour un **type** donné.

La seule réserve **STATIQUE** qu'on fait est un emplacement (une adresse statique pour le pointeur) qui va contenir l'adresse de la variable au moment de l'exécution (adresse **DYNAMIQUE** pour la **variable pointé**).

**Mais vous avez toujours dit qu'on ne peut pas faire de réserve pendant l'exécution.**

On ne **POUVAIT** pas.

Mais, **maintenant**, avec les pointeurs **ON PEUT**.



## Et comment ?

### Vous suivez.

Vous avez vu que chaque fois qu'on introduisait un nouveau type (entier, réel, tableau, enregistrement,...) on donnait les différentes actions permettant de manipuler ce type.

Et bien pour le type pointeur:

- On peut faire une **affectation** entre pointeurs :  $P \leftarrow Q$ ;  
à condition que P et Q pointent vers le même type de variable
- On peut **modifier** l'espace **pointé**:  $P^{\wedge} \leftarrow \text{<Expression>}$ ;  
où **<Expression>** est du même type que la variable **pointée**  
où encore un pointé du même type ( $Q^{\wedge}$ )
- On peut **Lire** ou **Ecrire** l'espace **pointé** : **Lire**( $P^{\wedge}$ ); **Ecrire**( $P^{\wedge}$ );  
si P pointe vers un enregistrement, il faut spécifier le nom du **champ** : **Lire**( $P^{\wedge}.\text{nom}$ );

-On peut **incrémenter** un pointeur :  $P \leftarrow P + 1$ ;  
mais dans ce cas la **nouvelle valeur** de **P** n'est pas **P+1**, mais on  
ajoute la **taille** de la variable **pointée**.

**Exemple:** Soit  $P: ^{\wedge}\text{reel}$ ;

Alors en supposant qu'un réel est représenté sur **4 octets**.

Si initialement **P=523120** on aura **P+1 = 523124**

En général on peut ajouter ou soustraire un entier d'un pointeur

$$P \leftarrow P + k; \quad \text{ou} \quad P \leftarrow P - k$$

mais toujours on tient compte de la taille de la variable pointé, donc  
ajouter ou soustraire :  $k * \text{Taille}$ .

**Attention:** La **soustraction** entre 2 pointeurs est un **entier**.  
L'**addition** entre pointeurs n'est **pas permise**.

Mais le plus **important** est :

- On peut **réserver** un espace mémoire **pendant** l'exécution. C'est ce qu'on appelle une **Allocation**. (Donc Allocation **DYNAMIQUE**).
- On peut aussi **Libérer** cet espace une fois qu'on n'aura plus besoin.

## Syntaxe

Soit la déclaration :  $P : ^\wedge\langle \text{TypeVar} \rangle;$

- Allocation d'espace : **Allouer(P);**
- Libération d'espace : **Liberer(P);**

### Exemple:

$P : ^\wedge \text{reel};$

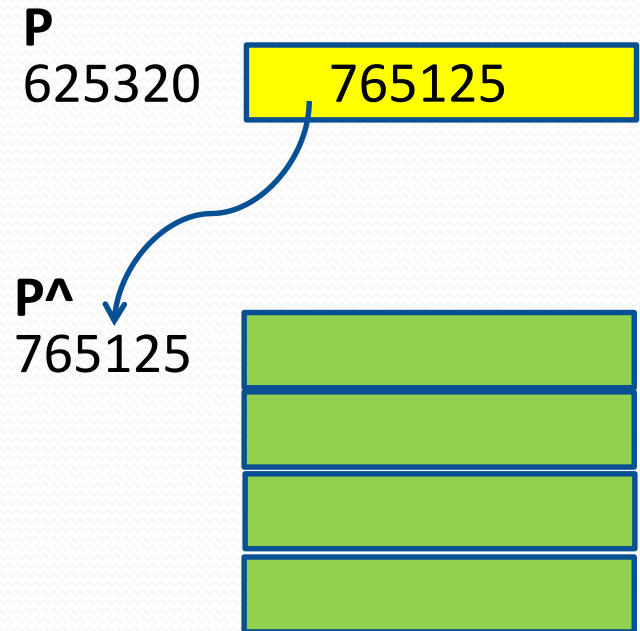
Cette déclaration permet de réserver un espace **statique vide** (ex : 625320) qui devra contenir **l'adresse** d'un réel.

Pendant l'exécution:

**Allouer(P);**

Cette action permet de :

- **Réserver** un espace pour un réel (Soit: 4 emplacements à partir d'une @ donnée par le système. **Exemple**: 765125)
- **Transmettre** l'adresse de cet emplacement (765125) vers l'emplacement de P (625320). Et donc on **crée un lien** entre le **pointeur P** et la donnée (**pointé**)  $P^\wedge$



Pendant l'exécution:

**Liberer(P);**

Cette action permet de :

**Libérer** l'espace réservé pour le réel (les 4 emplacements à partir de l'@ 765125)

**Couper** le lien entre le pointeur **P** et la donnée (pointé) **P^** (**qui n'existe plus**).

**Mais l'adresse de P existe toujours ! Et elle contient l'ancienne adresse du réel !**

**Effectivement**, cette action ne touche pas le pointeur, elle **concerne** l'espace **pointé** seulement (l'adresse n'est plus **occupée**, elle est déclarée **libre** par le système).

Pour ne pas avoir des erreurs de ce type, on utilise une constante spécifique « **Nil** » qui veut dire **vide**. Donc un pointeur qui contient **Nil**, ne pointe vers aucun espace.

**Conseil** : Il est conseillé d'affecter la constante **Nil** après la libération d'un espace pointé ( **P ← Nil ;** ).

625320

765125

765125

## Exemple

## Exécution

**Algorithme** Pointeurs;

**Var** P,Q : ^entier;

**Debut**

**Allouer**(P);

**Lire**(P^);

Q  $\leftarrow$  P;

Q^  $\leftarrow$  P^+5;

**Allouer**(Q);

Q^  $\leftarrow$  P^ ;

P  $\leftarrow$  Nil ;

**Ecrire**(Q^);

**Liberer**(Q);

**Fin.**

Réserver 2 emplacements mémoires (statiques) pour 2 pointeurs vers les entiers

@P

@x

@Q

@x

20

Réserver 1 emplacement mémoires (dynamique) pour 1 entier, soit à l'adresse @x

Transmettre l'adresse @x à P pour créer le lien entre le pointeur et le pointé

Lire une valeur et la mettre à l'adresse @x (pointé), soit cette valeur **20**

**Algorithme** Pointeurs;

**Var** P,Q : ^entier;

**Debut**

**Allouer**(P);

**Lire**(P^);

  Q ← P;

  Q^ ← P^+5;

**Allouer**(Q);

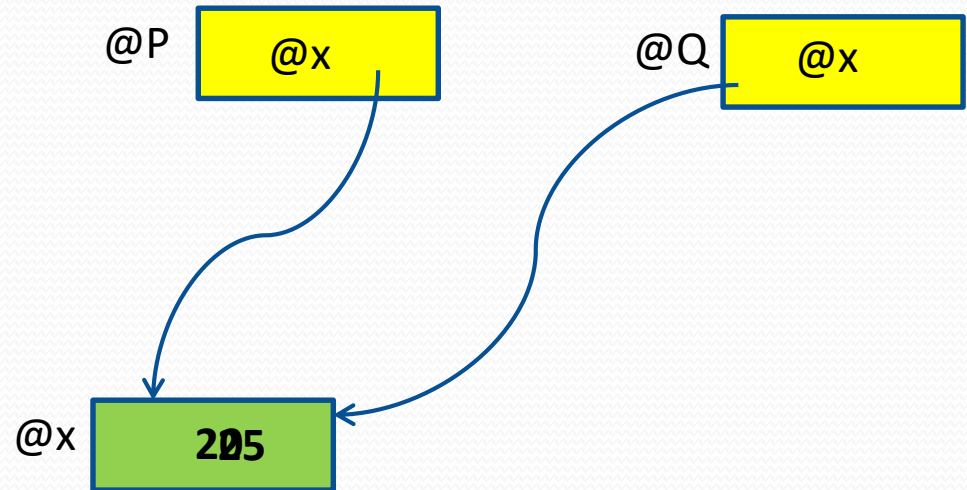
  Q^ ← P^ ;

  P ← Nil ;

**Ecrire**(Q^);

**Liberer**(Q);

**Fin.**



Transmettre l'adresse @x à Q pour créer le lien entre le pointeur et le pointé. **SANS FAIRE UNE ALLOCATION**. Donc P et Q pointent vers la même zone

Pointé de Q (@x) reçoit le pointé de P (toujours @x) +5, il devient 25.

**Algorithme** Pointeurs;

**Var** P,Q : ^entier;

**Debut**

**Allouer**(P);

**Lire**(P^);

  Q ← P;

  Q^ ← P^+5;

**Allouer**(Q);

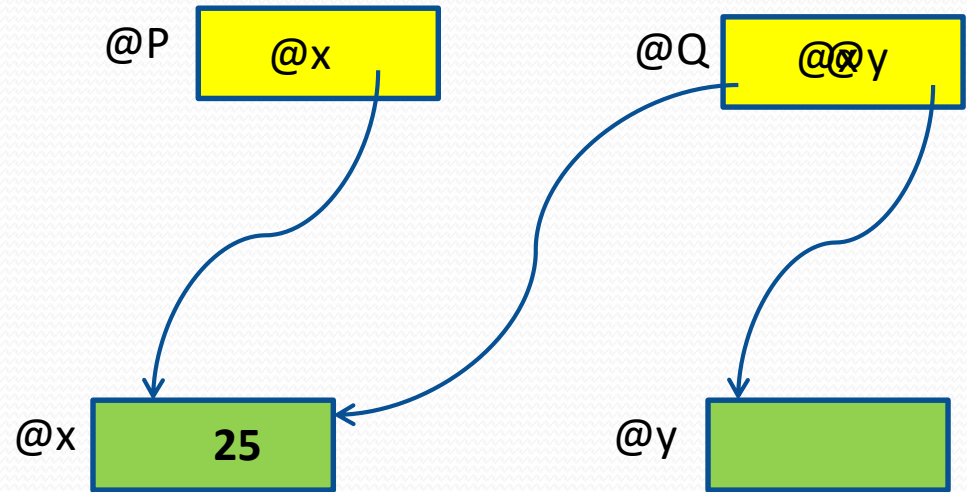
  Q^ ← P^ ;

  P ← Nil ;

**Ecrire**(Q^);

**Liberer**(Q);

**Fin.**

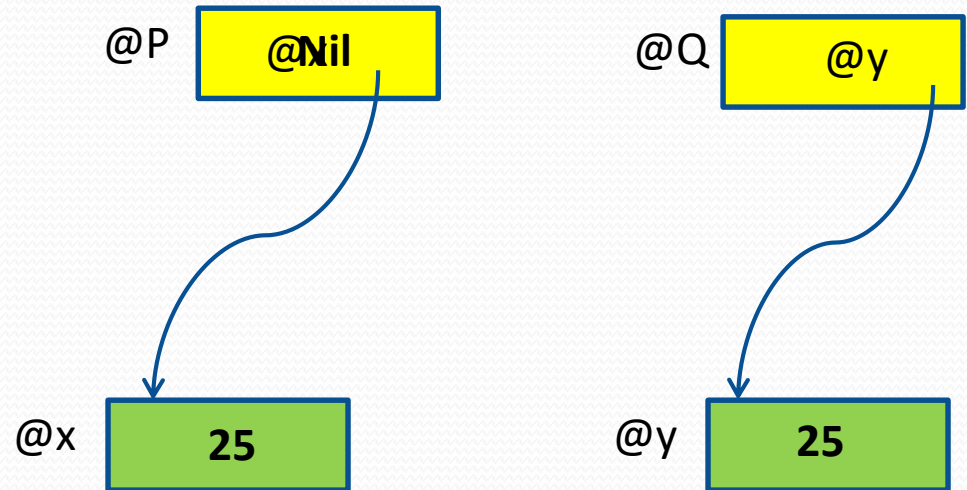


Réserver 1 emplacement mémoires (dynamique) pour 1 entier, soit à l'adresse @y

Transmettre l'adresse @y à Q pour créer le lien entre le pointeur et le pointé. Donc on coupe le lien entre Q et @x



```
Algorithme Pointeurs;  
Var P,Q : ^entier;  
Debut  
  Allouer(P);  
  Lire(P^);  
  Q ← P;  
  Q^ ← P^+5;  
  Allouer(Q);  
  Q^ ← P^ ;  
  P ← Nil ;  
  Ecrire(Q^);  
  Libérer(Q);  
Fin.
```



Le pointé de Q (@y) reçoit le pointé de P (@x). Donc on affecte @y par 25

P ne pointe plus vers @x, il ne pointe vers **rien (Nil)**. Donc on coupe le lien entre P et @x.

**Remarque importante:** L'espace mémoire d'adresse @x **existe** toujours, mais on a **perdu** son adresse dans le programme. Il restera ainsi **occupé** jusqu'à la fin du programme. Et on ne peut pas **l'exploiter**, car on ne la pas **libéré**. **IL NE FAUT JAMAIS FAIRE CETTE ERREUR**

**Algorithme** Pointeurs;

**Var** P,Q : ^entier;

**Debut**

**Allouer**(P);

**Lire**(P^);

  Q ← P;

  Q^ ← P^+5;

**Allouer**(Q);

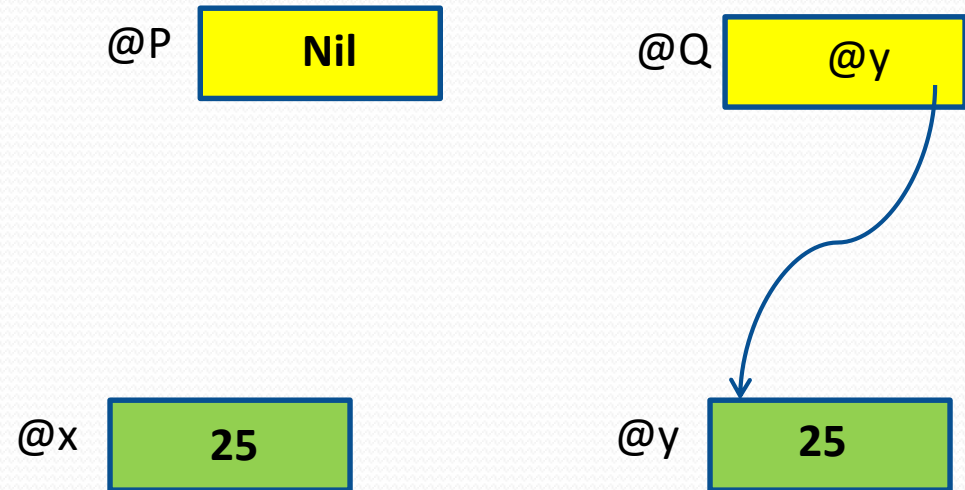
  Q^ ← P^ ;

  P ← Nil ;

**Ecrire**(Q^);

**Liberer**(Q);

**Fin.**



On affiche la valeur du pointé de Q (@y) qui est la valeur **25**

On libère l'espace pointé par Q (@y). Il ne fera plus partie du programme. Donc on coupe le lien entre Q et @y.

**Attention**, Q contient une adresse qui n'existe pas (@y).

**Conseil** : le mettre à **Nil** après avoir libérer le Q

Fin du programme, on libère tous les espaces : @P, @Q et même @x

# Exercice

Considérons le type enregistrement suivant :

**Type** TEtudiant = **Enregistrement**

Matricule : entier ;

Nom, Prenom : chaine [20] ;

Moyenne : réel ;

**Fin;**

Soit **T** un tableau de **N** étudiants ( $N \leq 100$ ). Ecrire un algorithme qui calcule le nombre d'étudiants admis.

- En utilisant une allocation statique.
- En utilisant une allocation dynamique.
- Quelle la différence entre les deux solutions.

# Solution



## 1- La solution statique est classique:

```
Algorithme SommeStatique;  
Type TEtudiant=Enregistrement  
    Matricule:entier;  
    Nom,Prenom:chaine[20];  
    Moyenne:réel;  
    Fin;  
Var T:Tableau[1..100] de TEtudiant;  
    I,N,S:entier;  
Debut  
    //lecture de la taille exacte  
    Ecrire('Donner la taille du tableau  $N \leq 100$ ');  
    Repeter Lire(N); Jusqu'à  $N > 0$  Et  $N \leq 100$ ;  
    //Lecture des éléments de T et calcul de S  
     $S \leftarrow 0$ ;  
    Pour  $I \leftarrow 1$  à N  
        Faire Avec T[I]  
            Faire  
                Lire(Matricule);  
                Lire(Nom,Prenom);  
                Lire(Moyenne);  
            Fait;  
            Si T[I].Moyenne  $\geq 10$  Alors  $S \leftarrow S + 1$  Fsi;  
    Fait;  
    Ecrire("Nombre d'étudiants admis= ",S);  
Fin.
```

# Solution

## 2- Solution Dynamique:

La solution dynamique consiste à utiliser des **pointeurs** pour les éléments du tableau (tableau de **pointeurs**). On va donc utiliser un tableau de pointeur vers le type Tetudiant. La réservation d'espace se fera pendant l'exécution. La déclaration du type Tetudiant reste inchangée.

```
Algorithme SommeDynamique;  
Type TEtudiant=Enregistrement  
    Matricule:entier;  
    Nom,Prenom:chaine[20];  
    Moyenne:réel;  
     Fin;  
    PTetud=^TEtudiant; //déclaration d'un pointeur  
Var T:Tableau[1..100] de PTetud; //tableau de pointeurs  
    I,N,S:entier;  
     Debut  
    //lecture de la taille exacte  
    Ecrire('Donner la taille du tableau N≤100');  
    Repeter Lire(N); Jusqu'à N>0 Et N≤100;  
    //Lecture des éléments de T et calcul de S  
    S←0;
```

```

Pour I←1 à N
Faire
    Allouer(T[I]); //allouer un espace
    Avec T[I]^ //variable dynamique
    Faire
        Lire(Matricule);
        Lire(Nom,Prenom);
        Lire(Moyenne);

    Fait;
    (*la référence à la variable dynamique se fait en ajoutant le
    symbole (^) à la variable pointeur*)
    Si T[I]^.Moyenne≥10 Alors S←S+1 Fsi;
Fait;
Ecrire('Nombre d''étudiants admis= ',S);
Fin.

```

### 3- Différence entre les deux solutions:

La différence entre les deux solutions est que dans la première (**statique**), on fait une réservation de **100 espaces** mémoires pour un enregistrement de type **TEtudiant** même si le nombre d'étudiants traités réellement (**N**) est inférieur à **100**.

Or dans la deuxième solution (**dynamique**), on fait une réservation d'espace mémoire au moment de l'exécution (**Allouer()**).

et donc on ne réserve que **N espaces** (exactement le nombre traité).

Il faut faire attention à la différence entre le pointeur (dans ce cas **T[I]** qui est une **variable statique**) et le pointé (dans ce cas **T[I]^** (suivi de ^)) qui est une **variable dynamique**.

Il faut noter que les espaces réservés pour les pointeurs (**100**) existent toujours (les espaces des pointeurs sont **statiques**). On **gagne** au niveau des espaces des pointés (qui sont **dynamiques**).

### Un peu de calcul ?!

Supposons que :

1 entier occupe **2** Octets.

1 réel occupe **4** Octets.

1 caractère occupe **1** Octet.

1 adresse occupe **2** Octets.

Dans ce cas la taille du type **TEtudiant** est :

**2** + **1**x20 + **1**x20 + **4** = **46** Octets. (**1** entier, **40** caractères , **1** réel)



Pour la **solution 1** (**statique**), on aura besoin de :

- 1 Tableau de 100 **TEtudiant** =  $100 \times 46 = 4600$  Octets.
- 3 entiers (I,N,S) =  $3 \times 2 = 6$  Octets.

Soit un Total = **4606** Octets.

Pour la **solution 2** (**dynamique**), on aura besoin de :

- 1 Tableau de 100 Pointeurs =  $100 \times 2 = 200$  Octets.
- N espaces Tetudiants =  $N \times 46 = 46N$  Octets.
- 3 entiers (I,N,S) =  $3 \times 2 = 6$  Octets.

Soit un Total = **46N + 206** Octets.

**Donc** on voit que l'espace nécessaire dépend de **N** (pendant l'exécution)

La **solution 2** sera **meilleure** lorsque : **46N+206 < 4606** (moins d'espace).

D'où : **N < (4606-206)/46** ce qui donne **N < 95.65**

Donc si le nombre d'étudiants traité est moins de **96** on aura toujours la **solution 2** qui est **meilleure**.

Ce n'est qu'à partir de **96** (très proche du cas statique) que la **solution 1** sera meilleure.

# Peut-on faire mieux ?

**Un peu mieux, OUI.**

On peut introduire ce qu'on appelle Un **Tableau Dynamique**

Pour ce (**nouveau type**), on considère le tableau comme une variable pointeur.

**Et c'est possible ?**

**Vous vous souvenez**, dans le cas des tableaux, on avait dit qu'on fait correspondre une seule adresse, c'est l'adresse du premier élément. Ensuite les adresses des autres éléments sont calculées par rapport à l'adresse de base.

**Pour un tableau Dynamique**, on fait la même chose, sauf que la réservation se fait pendant l'exécution. On réserve **N espaces contigus** (voisins) à partir d'une adresse qui sera l'adresse du premier élément. Un **tableau dynamique** se comporte exactement comme un tableau statique.

La déclaration d'un tableau dynamique se fait **sans préciser** la taille:

Exemple :

**T : Tableau de Tetudiant;** // et non pas de pointeur

le fait de ne pas donner la taille, veut dire que c'est dynamique.

L'allocation de l'espace du tableau se fait pendant l'exécution en **précisant** la taille :

**Allouer(T,N);**

**Attention** une fois alloué, on ne peut pas changer la taille d'un tableau (il devient statique). Si on veut changer la taille, il faut d'abord libérer l'espace, ensuite faire une nouvelle allocation.

Pour libérer l'espace du tableau on utilise la procédure **Liberer** :

**Liberer(T);**

# Applications des pointeurs

En plus de la gestion instantanée de la mémoire: possibilité **d'allouer** et de **libérer** de l'espace à tout moment pendant l'exécution, ce qui évite la **saturation** de la mémoire, on a vu une application directe des pointeurs et que vous **connaissiez bien**, mais peut être vous ne saviez pas que c'est une utilisation des pointeurs !

**Et c'est quoi ?**

**Et bien les Aps.**

On a vu que le **passage** des paramètres dans une **AP** peut se faire en deux modes:

1- **Par valeur** (ou en Entrée): dans ce cas on vous a dit qu'on travaille sur une **copie** de la variable (paramètre effectif). Et donc en sortant de l'AP, la valeur initiale de la variable reste inchangée même si elle subit des modifications à l'intérieur de l'AP.

**C'est exacte, oui. Et le deuxième mode c'était par référence ou en E/S.**

**Bien**, vous avez une bonne mémoire.

Le deuxième mode donc est le mode : par **référence** ou en **E/S** ou encore par **adresse**.

Dans ce cas on vous a dit qu'on travaille sur la **variable originale**, et toute modification touche directement la variable.

**Justement**, ceci est possible parce qu'on transmet **l'adresse** de la variable (donc un **pointeur**). Ainsi, à travers l'adresse, on travaille sur la copie originale de la variable.

**Autre application très importante des pointeurs:**

**Définition d'autres structures de données: Listes, Piles, Files, Arbre ...**

**Le cours prochain, justement, on va s'intéresser aux listes**

**Vous allez voir aussi (l'année prochaine) que la notion de pointeur est indispensable dans la POO (Programmation Orientée Objet).**

Merci!



[brbessaa@gmail.com](mailto:brbessaa@gmail.com)