Chapitre 9

Pointeurs

Inconvénients des variables statiques

Les variables « classiques », déclarées avant l'exécution d'un programme, pour ranger les valeurs nécessaires à ce programme, sont des *variables statiques*, c'est à dire que la place qui leur est réservée en mémoire est figée durant toute l'exécution du programme. Ceci a deux conséquences :

- Risque de manquer de place si la place réservée (par exemple le nombre d'éléments d'un tableau) est trop petite. Il faut alors que le programme prenne en charge le contrôle du débordement.
- Risque de risque de gaspiller de la place si la place réservée est beaucoup plus grande que celle qui est effectivement utilisée par le programme.

Un cahier, dont le nombre de pages est fixé a priori, fournit une bonne image d'une variable statique.

Introduction des variables dynamiques

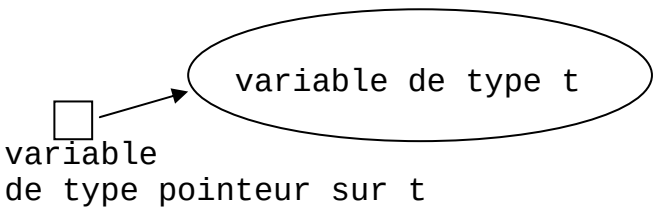
Les *variables dynamiques* vont permettre :

- de prendre la place en mémoire au fur et à mesure des besoins, celle-ci étant bien sûr limitée par la taille de la mémoire,
- de libérer de la place lorsqu'on n'en a plus besoin.

Un classeur dans lequel on peut ajouter ou retirer des pages est une bonne image d'une variable dynamique.

Outil : le pointeur

À partir d'un type `t` quelconque, on peut définir un type `pointeur_sur_t`. Les variables du type `pointeur_sur_t` contiendront, sous forme d'une adresse mémoire, un mode d'accès au contenu de la variable de type `t`. Celle-ci n'aura pas d'*identificateur*, mais sera accessible uniquement par l'intermédiaire de son *pointeur*.



Allocation / désallocation d'une zone de mémoire

On a défini un type `t`.
On déclare : `ptr : pointeur_sur_t ou ^t`.

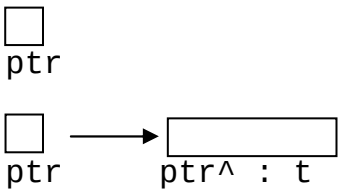
Création d'une variable dynamique de type t

`allouer(ptr)`

- réserve un emplacement mémoire de la taille correspondant au type `t`,
- met dans la variable `ptr` l'*adresse* de la zone mémoire qui a été réservée.

L'emplacement pointé par `ptr` sera accessible par `ptr^`.

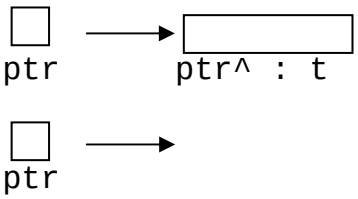
```
ptr : ^t

allouer(ptr)
```



Libération de la place occupée par une variable dynamique

`desallouer(ptr)`

- libère la place de la zone mémoire dont l'adresse est dans `ptr` (et la rend disponible pour l'allocation d'autres variables)
- laisse la valeur du pointeur en l'état (n'efface pas l'adresse qui est dans la variable pointeur).




```
desallouer(ptr)
```


Remarque : si on fait appel au pointeur désalloué, il renvoie une information qui n'a aucun sens.

Affectation entre pointeurs

Les pointeurs sont des variables particulières, puisque leurs valeurs sont des adresses mémoires. Ils peuvent néanmoins être impliqués dans des affectations au cours desquelles des adresses sont assignées aux pointeurs.

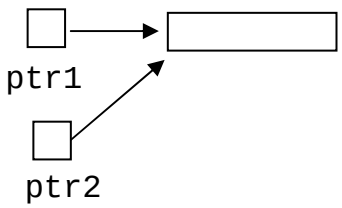
Pour « vider » un pointeur, c'est à dire annuler l'adresse qu'il contient, on lui affecte une valeur prédéfinie nommée *Nil* (ou *Null*). Attention, le fait de mettre *Nil* dans un pointeur ne libère pas l'emplacement sur lequel il pointait. L'emplacement devient irrécupérable car le lien vers cet emplacement a été coupé par la valeur *Nil*. Il faut désallouer avant d'affecter le pointeur avec *Nil*.

```
ptr ← Nil
```

Règle de bonne programmation : dès qu'on a désalloué un pointeur, il faut lui affecter la valeur *Nil*, pour qu'il ne conserve pas une adresse mémoire qui n'a plus d'existence physique.

On peut affecter un pointeur avec tout autre pointeur de même type. Après cette affectation, deux pointeurs désignent la même zone de mémoire.

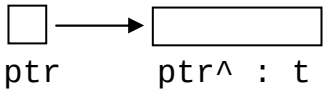
```
ptr1, ptr2 : pointeur sur t
allouer(ptr1)

ptr2  ← ptr1
```



Pour libérer la zone mémoire on désalloue `ptr1` ou (exclusif) `ptr2`, puis on met la valeur *`Nil`* dans `ptr1` et dans `ptr2`.

Accès à la variable pointée

Une fois la variable `ptr` déclarée et allouée, l'accès en écriture ou lecture à la variable pointée par `ptr` se fait avec l'identificateur `ptr^.` On peut appliquer à `ptr^` les même instructions qu'à une variable simple.

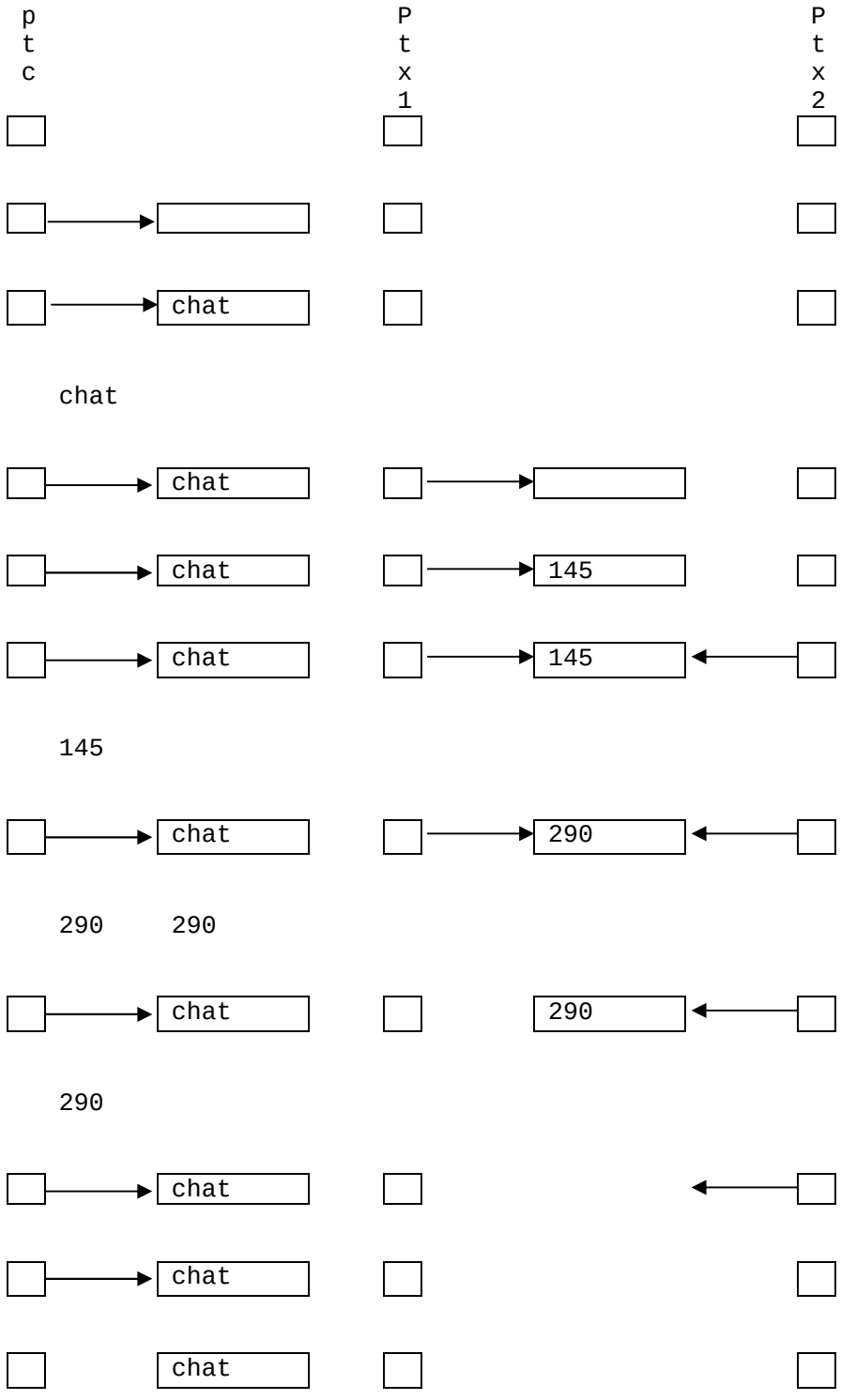


Affecter une valeur à la variable pointée :	`ptr^ ← <expression de type t>`
Lire une valeur de type `t` et la mettre dans la variable pointée :	`lire(ptr^)`
Afficher la valeur présente dans la zone pointée :	`écrire(ptr^)`

Un exemple pour comprendre

L'algorithme suivant n'a comme seul but que de faire comprendre la manipulation des pointeurs. À droite les schémas montrent l'état de la mémoire en plus de ce qui s'affiche à l'écran.

```
Variables
ptc : pointeur sur chaîne de caractères
ptx1, ptx2 : pointeur sur entier
```




```
Début
   allouer(ptc)
   ptc^ ← 'chat'
   écrire(ptc^)                    chat
   allouer(ptx1)
   lire(ptx1^)
   ptx2 ← ptx1
   écrire(ptx2^)                   145
   ptx2^ ← ptx1^ + ptx2^
   écrire(ptx1^, ptx2^)            290    290
   ptx1 ← Nil
   écrire(ptx2^)                   290
   désallouer(ptx2^)
   ptx2 ← Nil
   ptc ← Nil
Fin
```

Où est l'erreur ? : *on a coupé l'accès à la zone de mémoire sans la désallouer. Elle est irrécupérable*