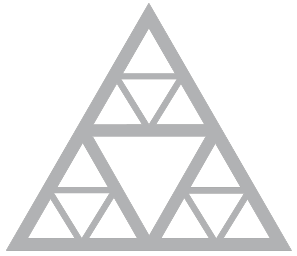
Introduction à la programmation en C++

Allocation dynamique

Nicolas Audebert

Vendredi 18 octobre 2018



Avant toute chose

Rendus de TP et des exercices

Les rendus se font sur Educnet.

1. Le code rendu **doit compiler**.
2. Le code rendu doit **être propre** (indentation, noms de variables clairs).
3. Le code rendu doit **être commenté** (réponses aux questions, fonctionnement du code).
4. Rassembler le code dans une seule archive (**Щńîĕ**, **ЩĝÄĝ**, **ЩĨÄĝЩæń**, etc.).

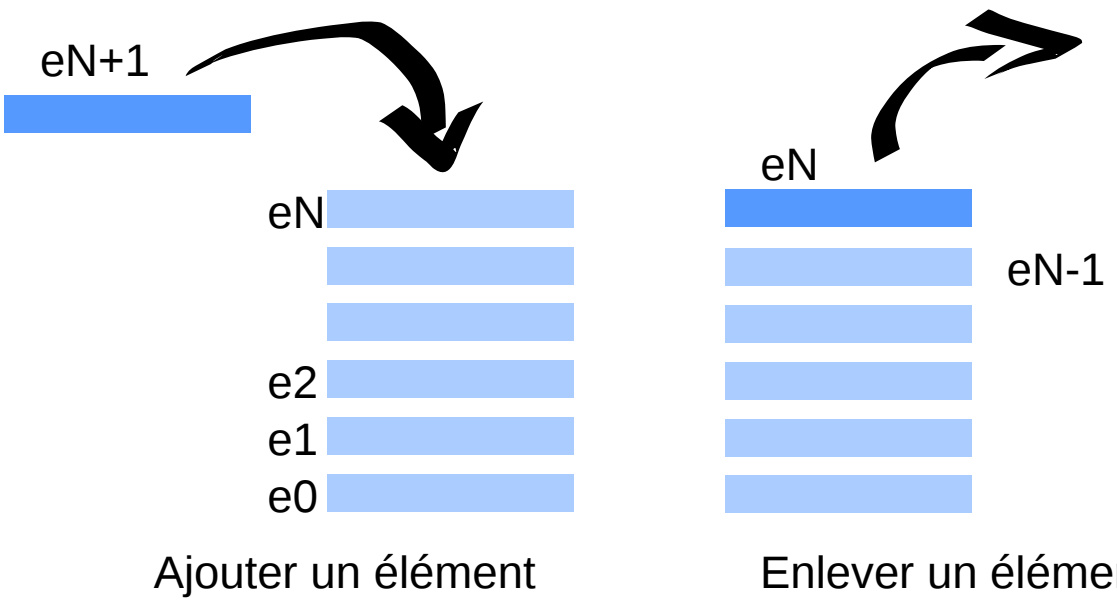
Plan de la séance

Pile des fonctions

Les appels aux fonctions sont gérés à l'aide d'une pile.

- **Entrer dans une fonction** : ajouter un élément à la pile
- **Sortir d'une fonction** : enlever un élément à la pile

La pile des fonctions permet de garder en mémoire l'ordre d'appel des fonctions. Chaque étage de la pile contient un **contexte d'exécution**. Ce mécanisme permet l'utilisation de **fonctions récursives**.



Exemple - Calcul de $n!$

Expression mathématique sous forme récursive

$$fact(n) = \begin{cases} n \quad fact(n \quad 1) , & \text{si } n > 0 \\ 0 , & \text{sinon.} \end{cases}$$

Implémentation récursive en C++

```
îćĨ åÄÑĨmîćĨ ćни
    îåmć f  бни
        ЯЯ ,Äġ ×Û ÐÄġÛ
        ĝÛĨĭĝć ЭЮ
    й ÛĀġÛ и
        ЯЯ @ĎĝĆĭĀÛ ×Û ĝÜÑĭĝĝÛćÑÛ
        ĝÛĨĭĝć ć П åÄÑĨmć ю ЭнЮ
    й
й
```

Tableaux dynamiques

Le tas

Le tas est une autre zone mémoire (différente de la pile) qui est accessible dynamiquement à la demande du programme.

```
îćĨ ĨÄîĀĀÛ ƎÛиЮ ЯЯ ĨÄîĀĀÛ ĔÄġ åĎĝÑÜĆÛćĨ ÑĎćġĨÄćĨÛ

îćĨП ĨÄÐ ćÛĹ îćĨкĨÄîĀĀÛлЮ ЯЯ ĝÜġÛĝкÛ ĀÄ ĔĀÄÑÛ ×Äćġ ĀÛ ĨÄġ
ЯЯ ĭĨîĀîġÄĨîĎć ×ĭ ĨÄÐĀÛÄĭ ÑĎĆĆÛ ĭć ĨÄÐĀÛÄĭ ÑĀÄġġîĜĭÛ
ЩЩЩ
×ÛĀÛĨÛкл ĨÄÐЮ ЯЯ ×ÜġÄĀĀĎĭÛ ĀÄ ĆÜĆĎîĝÛ ĎÑÑĭĔÜÛ ×Äćġ ĀÛ ĨÄġ
```

- La taille du tableau n'a pas à être connu au moment de la compilation,
- Le tableau peut changer de taille au cours du programme,
- Il ne faut pas oublier le `×ÛĀÛĨÛ кл ćĎĆ` .

Plan de la séance

Tableaux 2D

Les tableaux 2D de taille constante sont autorisés en C++.

```
ЯЯ 2ÜÑĀÄĝÄĨîĎć ×Эĭć ĨÄÐĀÛÄĭ Y2
×ĎĭÐĀÛ ĨÄÐY2кфлкЎлЮ
ЯЯ  ÑÑâġ Äĭľ ÜĀÜĆÛćĨġ
åĎĝмîćĨ îͨ6Ю îғфЮ îŧkнни
    åĎĝмîćĨ úͨ6Ю úғЎЮ úŧkни
        ĨÄÐY2кîлкúлͨ îПúЮ
        ЯЯ  ~~6_~Ie_ У ĔÄġ ×Û ЬĨÄÐY2кîФúлЬ
        ÑĎĭĨ ғғ ĨÄÐY2кîлкúл ғғ Ь ЬЮ
    й
    ÑĎĭĨ ғғ Ûć×ĀЮ
й
ЯЯ Ićî̃îÄĀîġÄĨîĎć
îćĨ ĨY2кЎлкЎлͨ ииϑФЎФЎйФиŸФфФѿййЮ
```

En fait, `îćĨ ĨY2кЎлкЎлЮ` est un tableau de tableaux : `ĨY2к6л` et `ĨY2кϑл` sont des tableaux de 3 cases.

Tableaux 2D et fonctions

On peut utiliser les tableaux 2D dans les fonctions, mais il faut en spécifier les dimensions dans la signature de la fonction :

```
кĎî× îćîĨмîćĨ ĨкΥлќΥлФ îćĨ кÄĀни
ЯЯ qÄġġÄæÛ îĆĔĀîÑîĨÛ ĔÄĝ ĝÜåÜĝÛćÑÛ
    åĎĝмîćĨ îˆ̣̈бЮ îғΥЮ îŧkни
        åĎĝмîćĨ úˆ̣̈бЮ úғΥЮ úŧkни
            Ĩкîлкúлˆ̣̈ кÄĀЮ
        й
    й
й

кĎî× åмни
    îćĨ ĨÄĐкΥлќΥлЮ
    îćîĨмĨÄĐФ бнЮ ЯЯ ÄĔĔÛĀ ×Û ĀÄ åĎćÑĨîĎć ġĭĝ ĀÄ кÄĝîÄĐĀÛ ĨÄĐ
й
```

Limitations

Fonctions génériques

Cas 1D : il est possible de faire des fonctions génériques

```
кĎî× îćîĨмîćĨ Ĩклф îćĨ ĨÄîĀĀÛф îćĨ кÄĀни
    åĎĝмîćĨ î  6Ю îғĨÄîĀĀÛЮ îkkни
        Ĩкîл  кÄĀЮ
    й
й
```

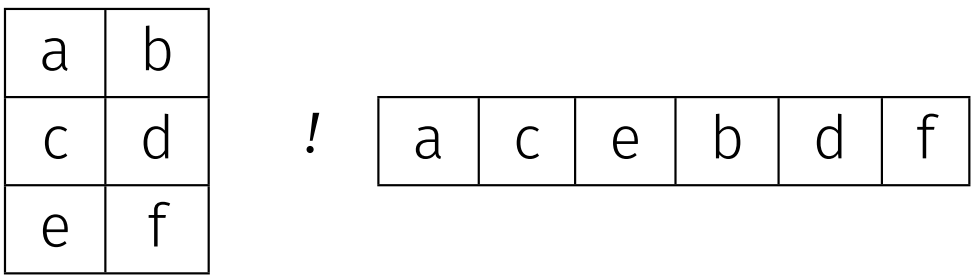
Cas 2D : ce n'est pas possible

```
кĎî× îćîĨмîćĨ ĨклклФ îćĨ ĝĎĹġф îćĨ ÑĎĀġф îćĨ кÄĀниЩЩЩЙ ЯЯ6tt6¤t
```

! Réécriture de code ? une fonction pour chaque taille de tableau ?

Solution

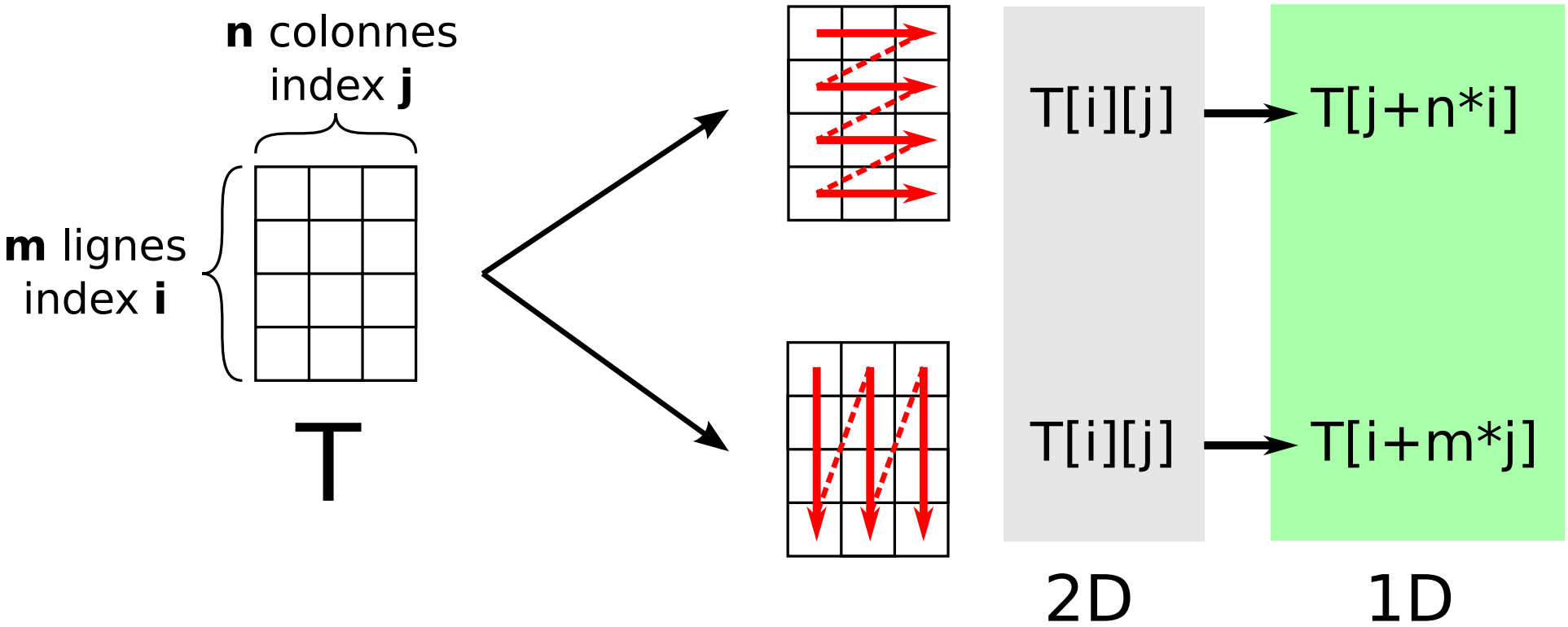
On utilise des toujours des tableaux à 1 dimension.



Cette solution permet de gérer autant de dimension qu'on le souhaite.

Parcourir un tableau 2D ! 1D

On utilise soit le **parcours en lignes**, soit le **parcours en colonnes**.



Utilisation dans les fonctions

Il est désormais possible d'utiliser des fonctions génériques.

```
×ĎĭÐĀÛ åîĀĀм×ĎĭÐĀÛ ĆÄĨклΦ îćĨ ĝĎĹġΦ îćĨ ÑĎĀġΦ ×ĎĭÐĀÛ кÄĀни
    åĎĝмîćĨ îͨ͛бЮ îғĝĎĹġЮ îŧŧни
        åĎĝмîćĨ úͨ͛бЮ úғÑĎĀġЮ úŧŧни
            ĆÄĨкúŧÑĎĀġПîлͨ͛ кÄĀЮ
        й
    й
й

кĎî× ĚĝĎ×бĆÄĨбкÛÑм×ĎĭÐĀÛ ĆÄĨклΦ îćĨ ĝĎĹġΦ îćĨ ÑĎĀġΦ ×ĎĭÐĀÛ кÛÑклΦ
                                                    ×ĎĭÐĀÛ ġĎĀклни
    åĎĝмîćĨ îͨ͛бЮ îғĝĎĹġЮ îŧŧни
        ġĎĀкîлͨ͛ бЮ
        åĎĝмîćĨ úͨ͛бЮ úғÑĎĀġЮ úŧŧни
            ġĎĀкîл ŧͨ͛ ĆÄĨкúŧÑĎĀġПîлПкÛÑкúлЮ
        й
    й
й
```

Plan de la séance

Tableaux de taille variable

Allocation dynamique et tableaux 2D

Pas de possibilité de faire des tableaux 2D avec allocation dynamique (tableaux de taille variable).

Solution

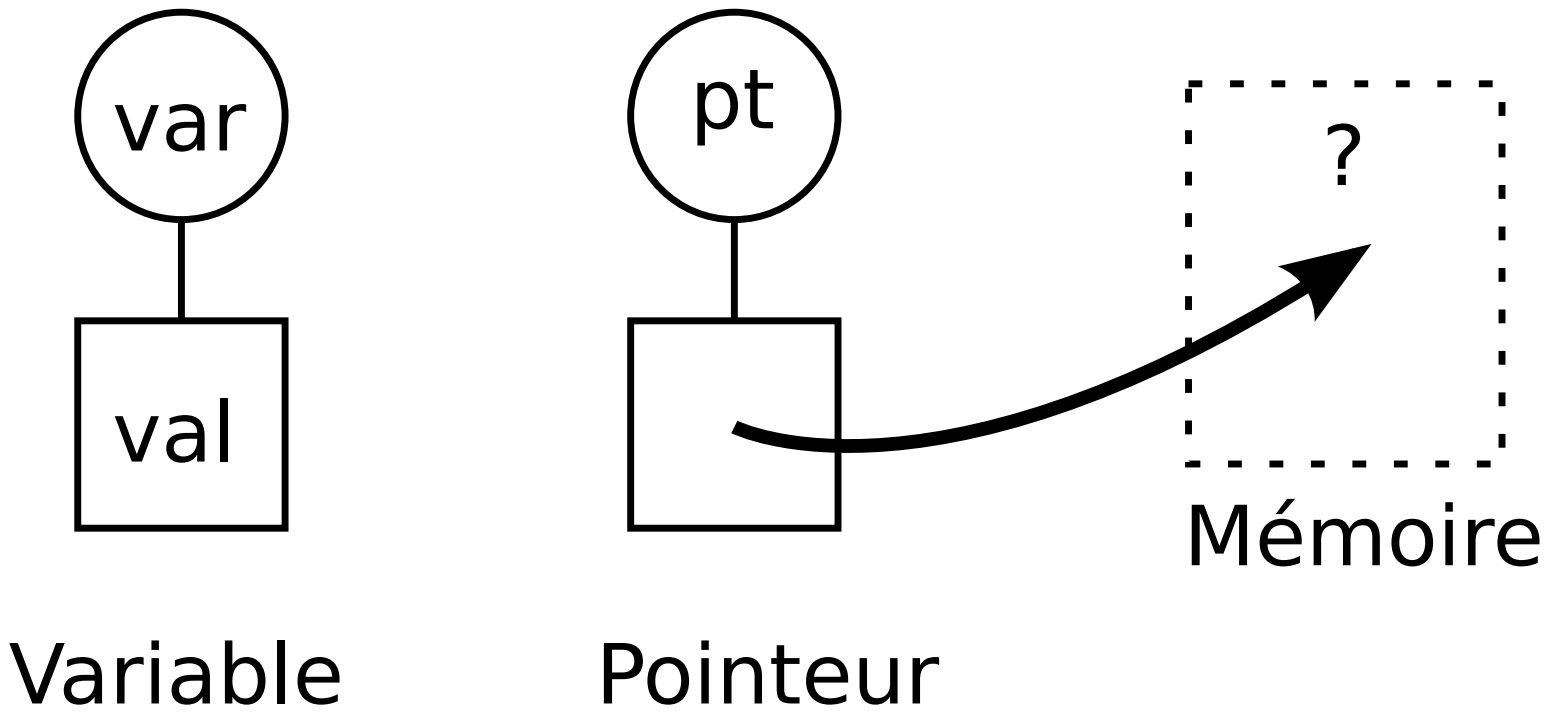
On fait des tableaux 1D, comme précédemment.

```
îćĨ Ć   ЩЩЩ Ю
îćĨ ć   ЩЩЩ Ю
×ĎĭÐĀÛП    ćÛĹ ×ĎĭÐĀÛкĆПćлЮ
×ĎĭÐĀÛП Ľ  ćÛĹ ×ĎĭÐĀÛкĆлЮ
×ĎĭÐĀÛП Ŀ  ćÛĹ ×ĎĭÐĀÛкćлЮ
ЩЩЩ
кĎî× ĔĝĎ×бĆÄĨбкÛÑм ФĆФĆФĽФĿнЮ
ЩЩЩ
×ÛĀÛĨÛкл  Ю
×ÛĀÛĨÛкл ĽЮ
×ÛĀÛĨÛкл ĿЮ
```

Pointeurs

Définition

Un **pointeur** est une variable qui stocke une adresse vers une zone mémoire (tableau ou variable) dans la pile ou dans le **tas**.



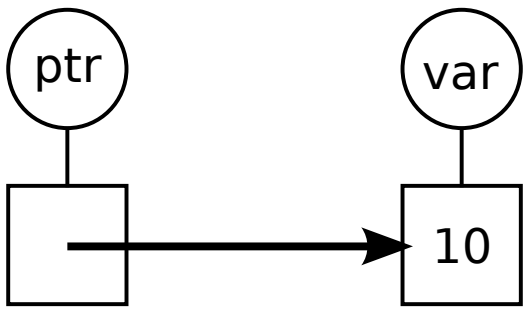
Déclarer un pointeur

On utilise le caractère **Π**.

```
îćĨΠ ĚĨĝЮ ЯЯ ĭć ĚĎîćĨÛĭĝ κÛĝġ ĭć ÛćĨîÛĝ
```

Pour récupérer l'adresse d'une variable on utilise le **W**

```
îćĨΠ ĚĨĝЮ ЯЯ ĭć ĚĎîćĨÛĭĝ κÛĝġ ĭć ÛćĨîÛĝ
îćĨ ĨÛġĨ ϑϭЮ
ĚĨĝ WĨÛġĨЮ ЯЯ ĀÛ ĚĎîćĨÛĭĝ ĝÛ×îĝîæÛ κÛĝġ ĨÛġĨ
```



Pointeurs et mémoire

L'intérêt d'utiliser des pointeurs avec des variables classiques est limité.

Des pointeurs pour le tas

Les pointeurs sont la porte d'entrée vers le tas (la mémoire de l'ordinateur).

- Créer une variable dans le tas : **ćÛĹ**
- Supprimer une variable dans le tas : **×ÛĀÛĨÛ**

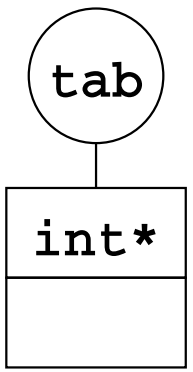
Pointeurs et tableaux

```
double* tab;
int n=5;
tab = new double[n];

for(int i=0; i<n; i++){
    tab[i] = 2*i;
}

delete[] tab;
```

La pile



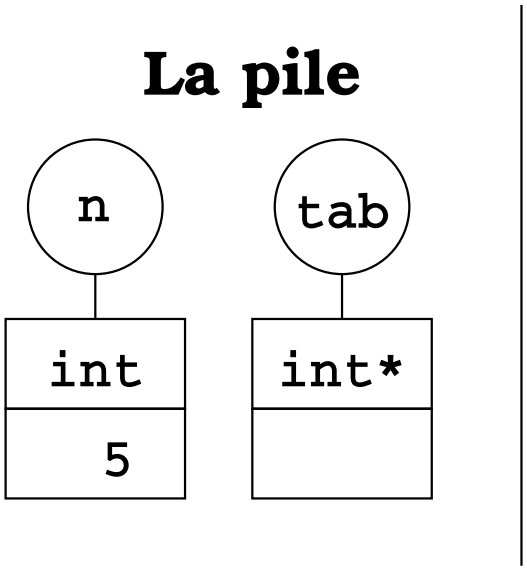
Le tas

Pointeurs et tableaux

```
double* tab;
int n=5;
tab = new double[n];

for(int i=0; i<n; i++){
    tab[i] = 2*i;
}

delete[] tab;
```

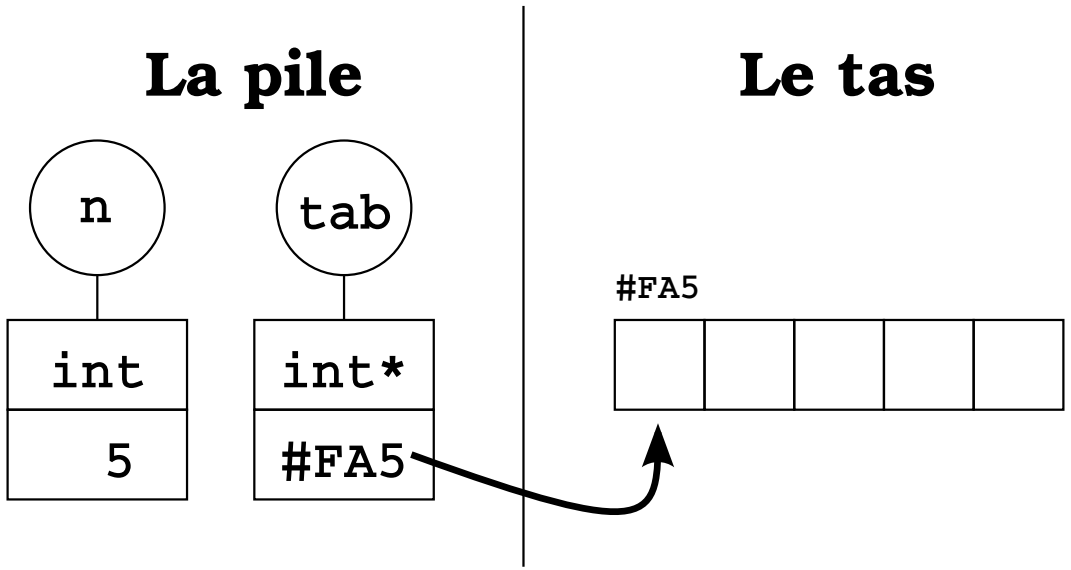


Pointeurs et tableaux

```
double* tab;
int n=5;
tab = new double[n];

for(int i=0; i<n; i++){
    tab[i] = 2*i;
}

delete[] tab;
```

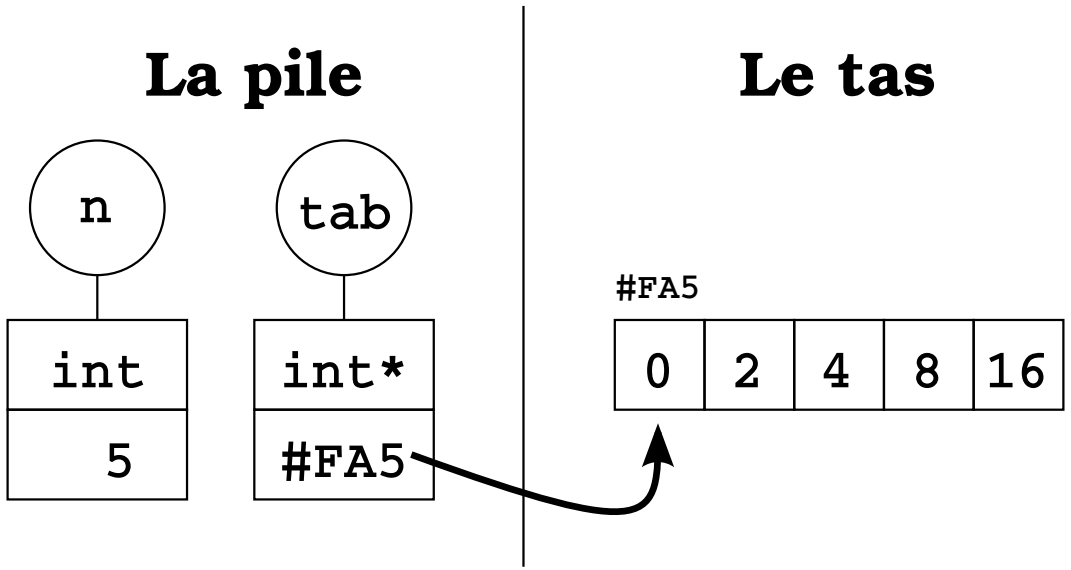


Pointeurs et tableaux

```
double* tab;
int n=5;
tab = new double[n];

for(int i=0; i<n; i++){
    tab[i] = 2*i;
}

delete[] tab;
```

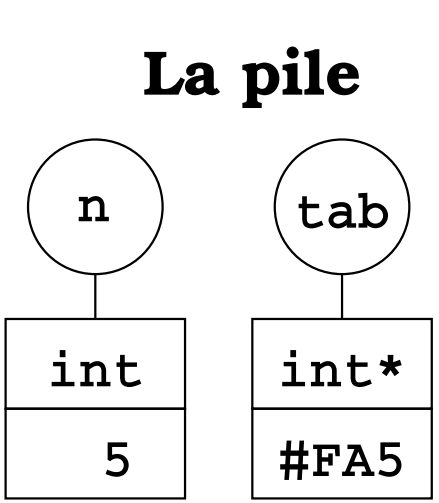


Pointeurs et tableaux

```
double* tab;
int n=5;
tab = new double[n];

for(int i=0; i<n; i++){
    tab[i] = 2*i;
}

delete[] tab;
```

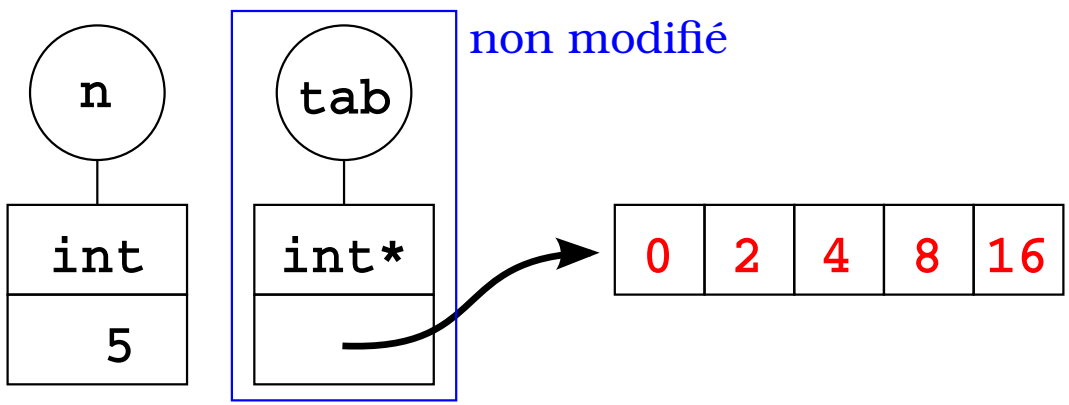


Modifier la variable / tableau désigné par le pointeurs

- Pas besoin de passage par référence : on ne modifie pas le pointeur (l'adresse), seulement les valeurs stockées dans la zone de la mémoire désignées par le pointeur.
- On peut utiliser les fonctions créées pour les tableaux statiques.

```
void fill(double* tab, int n){
  for(int i=0; i<n; i++)
    tab[i] = 2*i;
}

double* t;
int taille=5;
t = new double[taille];
fill(t,taille);
delete[] tab;
```

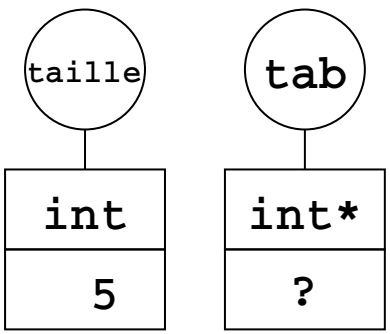


Modifier le pointeur

- Il faut faire un passage par référence : on modifie l'adresse stockée par le pointeur.

```
void create(double* tab, int n){
  tab = new double[n];
  cout << tab << endl;
}

double* t;
int taille=5;
create(t,taille);
cout << t << endl;
delete[] tab;
```

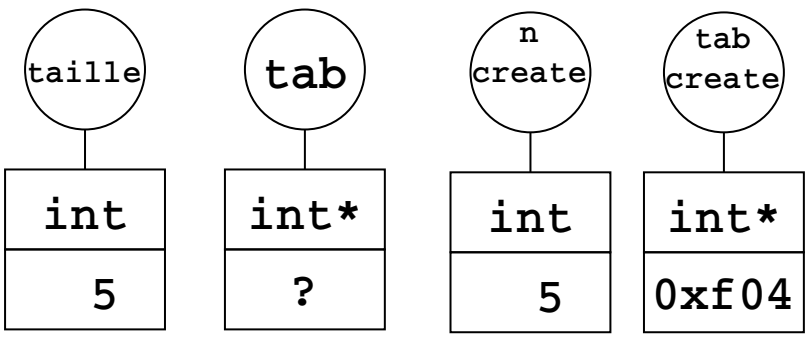


Modifier le pointeur

- Il faut faire un passage par référence : on modifie l'adresse stockée par le pointeur.

```
void create(double* tab, int n){
  tab = new double[n];
  cout << tab << endl;
}

double* t;
int taille=5;
create(t,taille);
cout << t << endl;
delete[] tab;
```

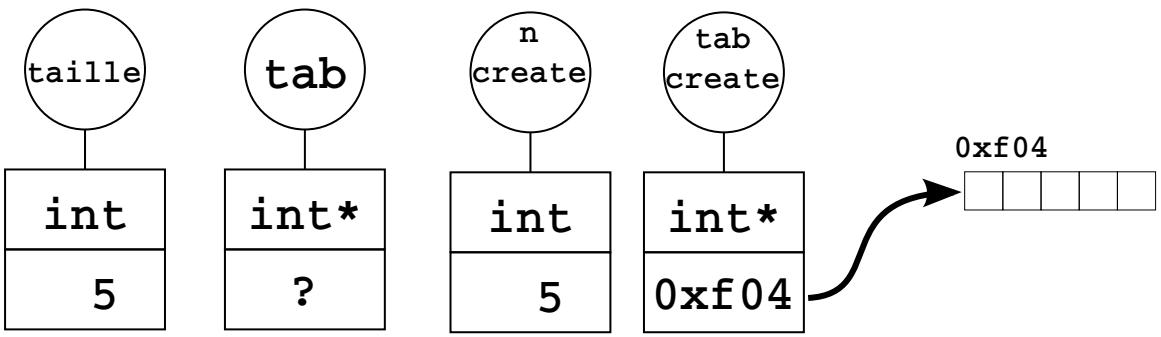


Modifier le pointeur

- Il faut faire un passage par référence : on modifie l'adresse stockée par le pointeur.

```
void create(double* tab, int n){
  tab = new double[n];
  cout << tab << endl;
}

double* t;
int taille=5;
create(t,taille);
cout << t << endl;
delete[] tab;
```

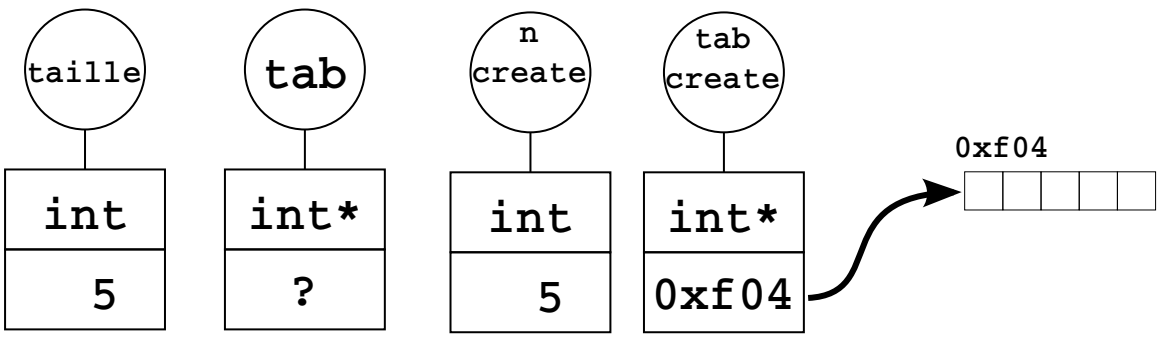


Modifier le pointeur

- Il faut faire un passage par référence : on modifie l'adresse stockée par le pointeur.

```
void create(double* tab, int n){
  tab = new double[n];
  cout << tab << endl;
}                              --> 0xf04

double* t;
int taille=5;
create(t,taille);
cout << t << endl;
delete[] tab;
```

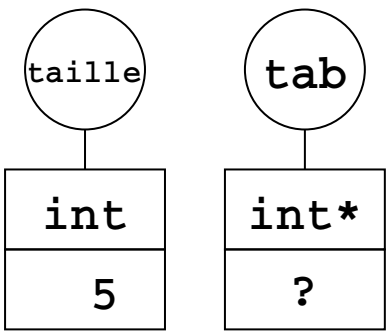


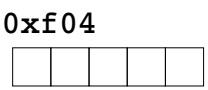
Modifier le pointeur

- Il faut faire un passage par référence : on modifie l'adresse stockée par le pointeur.

```
void create(double* tab, int n){
  tab = new double[n];
  cout << tab << endl;
}                          --> 0xf04

double* t;
int taille=5;
create(t,taille);
cout << t << endl;
delete[] tab;
```



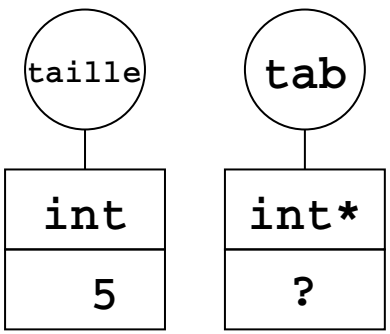


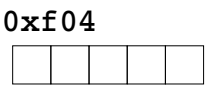
Modifier le pointeur

- Il faut faire un passage par référence : on modifie l'adresse stockée par le pointeur.

```
void create(double* tab, int n){
  tab = new double[n];
  cout << tab << endl;
}                          --> 0xf04

double* t;
int taille=5;
create(t,taille);
cout << t << endl;         --> ?
delete[] tab;
```



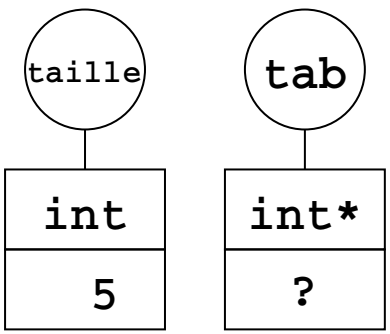


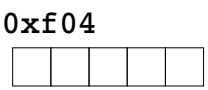
Modifier le pointeur

- Il faut faire un passage par référence : on modifie l'adresse stockée par le pointeur.

```
void create(double* tab, int n){
  tab = new double[n];
  cout << tab << endl;
}                          --> 0xf04

double* t;
int taille=5;
create(t,taille);
cout << t << endl;      --> ?
delete[] tab;  --> ERREUR non alloué
```





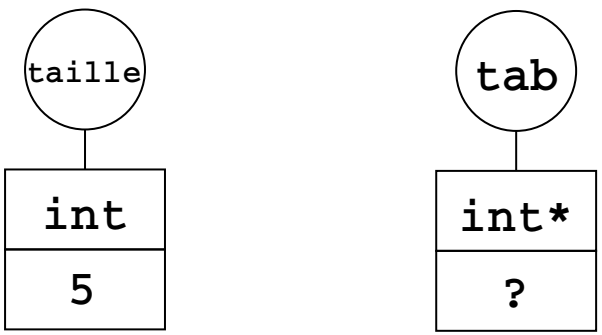
ERREUR FUITE DE MÉMOIRE

Modifier le pointeur

- Il faut faire un passage par référence : on modifie l'adresse stockée par le pointeur.

```
void create(double* &tab, int n){
  tab = new double[n];
  cout << tab << endl;
}

double* t;
int taille=5;
create(t,taille);
cout << t << endl;
delete[] tab;
```

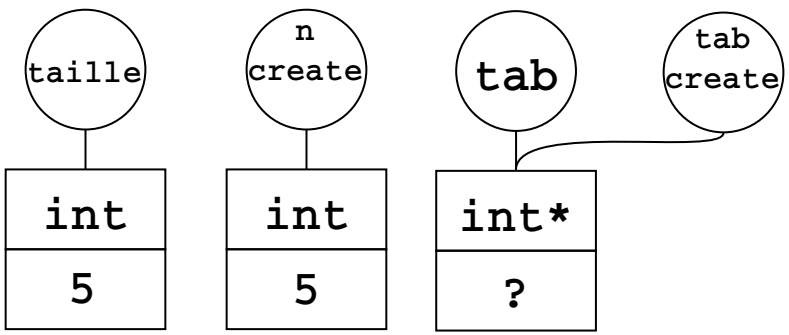


Modifier le pointeur

- Il faut faire un passage par référence : on modifie l'adresse stockée par le pointeur.

```
void create(double* &tab, int n){
  tab = new double[n];
  cout << tab << endl;
}

double* t;
int taille=5;
create(t,taille);
cout << t << endl;
delete[] tab;
```

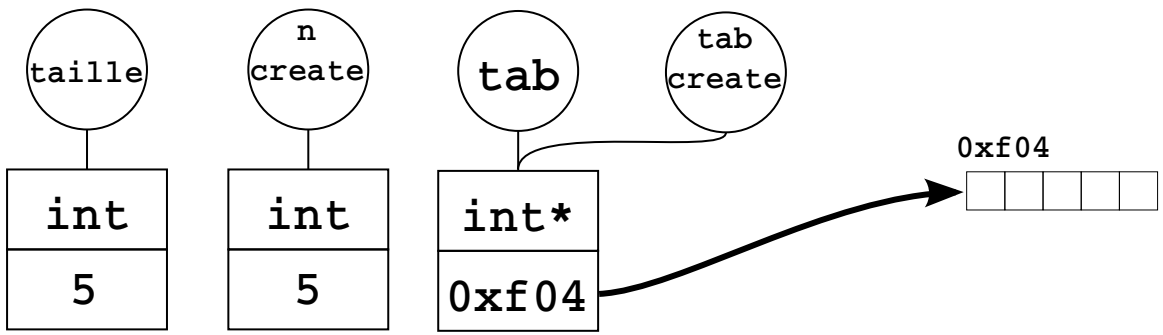


Modifier le pointeur

- Il faut faire un passage par référence : on modifie l'adresse stockée par le pointeur.

```
void create(double* &tab, int n){
  tab = new double[n];
  cout << tab << endl;
}

double* t;
int taille=5;
create(t,taille);
cout << t << endl;
delete[] tab;
```

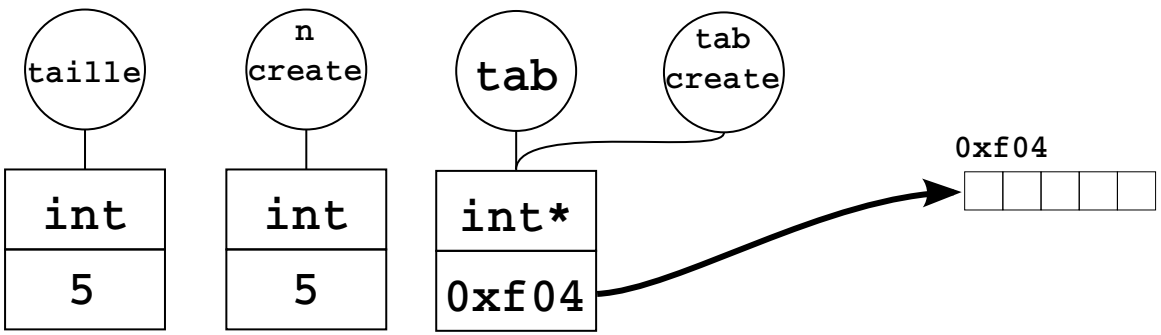


Modifier le pointeur

- Il faut faire un passage par référence : on modifie l'adresse stockée par le pointeur.

```
void create(double* &tab, int n){
  tab = new double[n];
  cout << tab << endl;
}                          --> 0xf04

double* t;
int taille=5;
create(t,taille);
cout << t << endl;
delete[] tab;
```



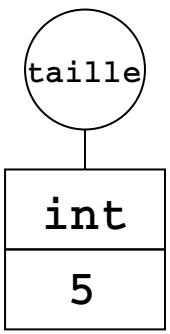
Modifier le pointeur

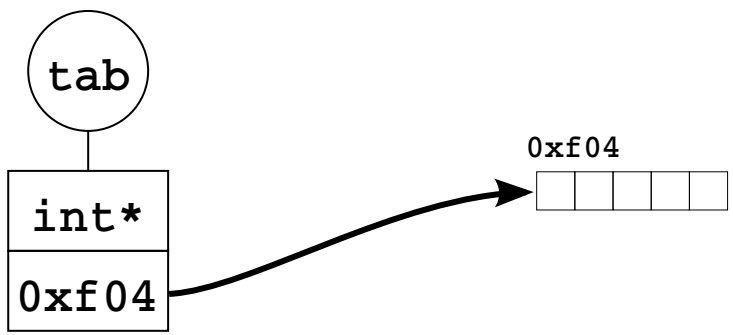
- Il faut faire un passage par référence : on modifie l'adresse stockée par le pointeur.

```
void create(double* &tab, int n){
  tab = new double[n];
  cout << tab << endl;
}

double* t;
int taille=5;
create(t,taille);
cout << t << endl;
delete[] tab;
```

--> 0xf04



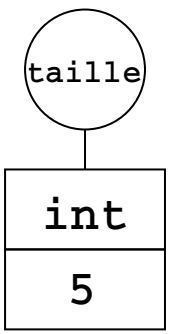


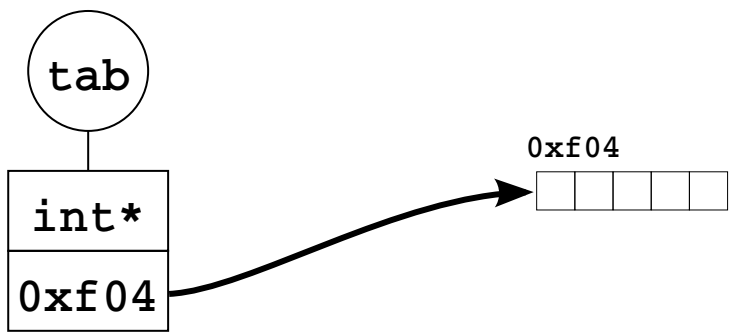
Modifier le pointeur

- Il faut faire un passage par référence : on modifie l'adresse stockée par le pointeur.

```
void create(double* &tab, int n){
  tab = new double[n];
  cout << tab << endl;
}

double* t;
int taille=5;
create(t,taille);
cout << t << endl;  --> 0xf04
delete[] tab;
```



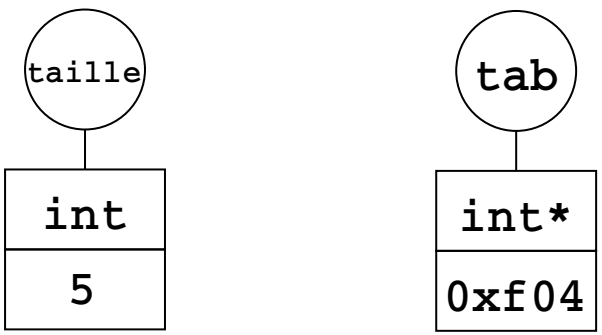


Modifier le pointeur

- Il faut faire un passage par référence : on modifie l'adresse stockée par le pointeur.

```
void create(double* &tab, int n){
  tab = new double[n];
  cout << tab << endl;
}

double* t;
int taille=5;
create(t,taille);
cout << t << endl;  --> 0xf04
delete[] tab;
```



Egalité de pointeurs

L'égalité de pointeurs est autorisée.

Attention

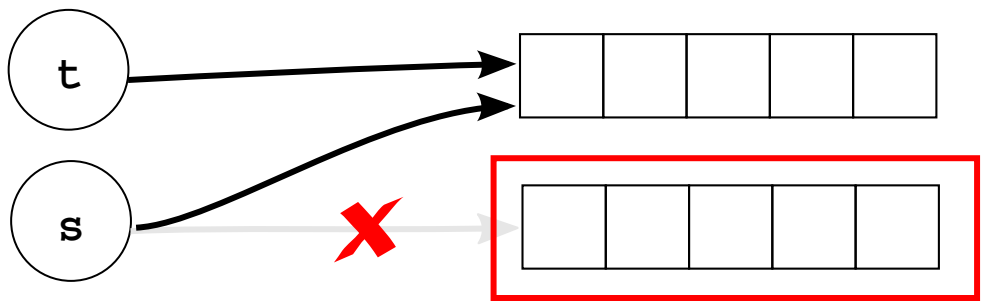
- Il y a des risques de fuite de mémoire
- Deux pointeurs égaux renvoient au même espace mémoire
- Il n'y a pas création d'un nouveau tableau

```
double* t,s;
int n=5;
t = new double[n];
s = new double[n];

s = t;

...
```

--> Fuite de mémoire



--> Fuite de mémoire

Egalité de pointeurs

L'égalité de pointeurs est autorisée.

Attention

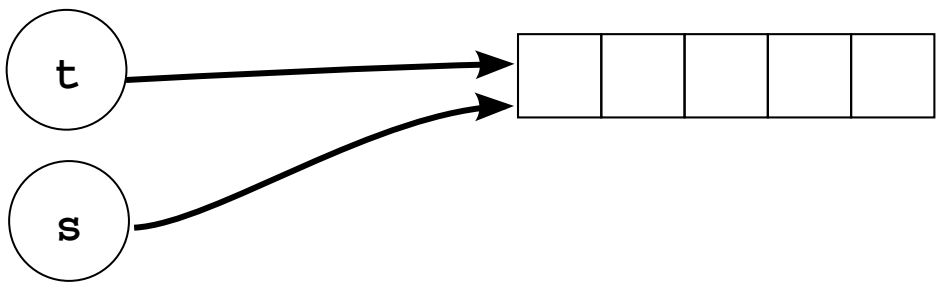
- Il y a des risques de fuite de mémoire
- Deux pointeurs égaux renvoient au même espace mémoire
- Il n'y a pas création d'un nouveau tableau

```
double* t,s;
int n=5;
t = new double[n];

s = t;

delete[] t;
delete[] s;
```

--> double délétion



Egalité de pointeurs

L'égalité de pointeurs est autorisée.

Attention

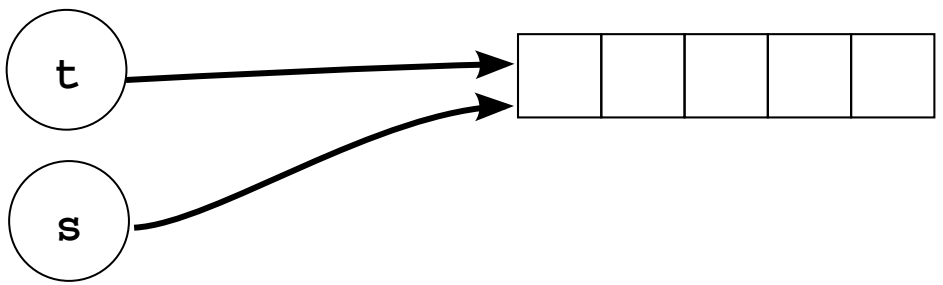
- Il y a des risques de fuite de mémoire
- Deux pointeurs égaux renvoient au même espace mémoire
- Il n'y a pas création d'un nouveau tableau

```
double* t,s;
int n=5;
t = new double[n];

s = t;

delete[] t; // ou delete[] s
```

--> OK



Egalité de pointeurs 2

Pour copier un tableau, il faut le faire terme à terme.

```
×ĎĭĐĀÛП ĨΦġЮ
îćĨ ć   ϑϬϬЮ
Ĩ   ćÛĹ ×ĎĭĐĀÛкćлЮ

ЩЩЩ

ЯЯ ÑĎĚîÛ ×ĭ ĨÄĐĀÛÄĭ
ġ   ćÛĹ ×ĎĭĐĀÛкćлЮ ЯЯ ÄĀĀĎÑÄĨîĎć ×Û ĀÄ ĆÛĆĎîĝÛ
åĎĝмîćĨ î   ϬЮ îғćЮ îkkни
    ġкîл   ĨкîлЮ ЯЯ ĝÛÑĎĚîÛ ĨÛĝĆÛ Ä ĨÛĝĆÛ
й
ЩЩЩ
×ÛĀÛĨÛкл ĨЮ ЯЯ ĀîĐÛĝÄĨîĎć ĨÄĐĀÛÄĭ Ĩ
×ÛĀÛĨÛкл ġЮ ЯЯ ĀîĐÛĝÄĨîĎć ĨÄĐĀÛÄĭ ġ
```

Plan de la séance

Des tableaux dans des structures

Il est possible d'utiliser des tableaux dynamiques dans les structures.

Attention

Surtout pas de tableaux statiques.

```
ġĨĝĭÑĨ ¯ÛÑĨи
    îćĨ ĨÄîĀĀÛЮ ЯЯ ĀÄ ĨÄîĀĀÛ
    ×ĎĭÐĀÛП ĨЮ ЯЯ ĀÛ ĨÄÐĀÛÄĭΦ ćÛ ĚÄġ ÄĀĀĎĭÛĝ
              ЯЯ ×Äćġ ĀÄ ×ÛÑĀÄĝÄĨîĎć ×Û ĀÄ ġĨĝĭÑĨĭĝÛ
йЮ
```

Structures et allocation dynamique

```
ЯЯ ¯ÛÑĨЩë
ġĨĝĭÑĨ ¯ÛÑĨи
    îćĨ ćЮ ЯЯ ĨÄîĀĀÛ
    ×ĎĭÐĀÛП ĨЮ ЯЯ ĨÄÐĀÛÄĭ
йЮ

кĎî× îćîĨм¯ÛÑĨw кнЮ

кĎî× ÑĝÛÛм¯ÛÑĨw кФ îćĨ ćнЮ

кĎî× ×ÛĨĝĭîĨм¯ÛÑĨw кнЮ

кĎî× ĝÛĆĔĀîĨм¯ÛÑĨw кФ ×ĎĭÐĀÛ кÄĀнЮ

кĎî× ÑĎĔîÛм¯ÛÑĨw кФ ¯ÛÑĨ ĎнЮ

¯ÛÑĨ ĎĔÛĝÄĨĎĝkм¯ÛÑĨ кЭФ ¯ÛÑĨ кУнЮ
```

Structures et allocation dynamique

```
ЯЯ ¯ÛÑĨЩё
ġĨĝĭÑĨ ¯ÛÑĨи
    îćĨ ćЮ ЯЯ ĨÄîĀĀÛ
    ×ĎĭÐĀÛП ĨЮ ЯЯ ĨÄÐĀÛÄĭ
йЮ

кĎî× îćîĨм¯ÛÑĨw кнЮ

кĎî× ÑĝÛÛм¯ÛÑĨw кФ îćĨ ćнЮ

кĎî× ×ÛĨĝĭîĨм¯ÛÑĨw кнЮ

кĎî× ĝÛĆĔĀîĨм¯ÛÑĨw кФ ×ĎĭÐĀÛ кÄĀнЮ

кĎî× ÑĎĔîÛм¯ÛÑĨw кФ ¯ÛÑĨ ĎнЮ

¯ÛÑĨ ĎĔÛĝÄĨĎĝkм¯ÛÑĨ кЭФ ¯ÛÑĨ кУнЮ
```

```
ЯЯ ¯ÛÑĨЩÑĔĔ
ШîćÑĀĭ×Û Ь¯ÛÑĨЩёЬ
кĎî× îćîĨм¯ÛÑĨw кни
    кЩć:: бЮ
й
кĎî× ÑĝÛÛм¯ÛÑĨw кФ îćĨ ćни
    Äġġ ÛĝĨмć Ь бнЮ
    кЩć:: ćЮ
    кЩĨ:: ćÛĹ ×ĎĭÐĀÛккЩćлЮ
й
кĎî× ×ÛĨĝĭîĨм¯ÛÑĨw кни
    îåмкЩĨÄîĀĀÛ Ь бни
        кЩĨÄîĀĀÛ:: бЮ
        ×ÛĀÛĨÛкл кЩĨЮ
    й
й
кĎî× ĝÛĆĔĀîĨм¯ÛÑĨw кФ ×ĎĭÐĀÛ кÄĀни
    åĎĝмîćĨ î:: бЮ îғкЩćЮ îkkни
        кЩĨкîл:: кÄĀЮ
    й
й
```

Structures et allocation dynamique

```
яя ¯ûññщё
ġĩĝĭññ ¯ûññи
    îćĩ ćю яя ĩäîāāû
    ×ďĭđāûп ĩю яя ĩäđāûäĭ
йю

кďî× îćîĩм¯ûññw кню

кďî× ñĝûûм¯ûññw кф îćĩ ćню

кďî× ×ûĩĝĭîĩм¯ûññw кню

кďî× ĝûćĕāîĩм¯ûññw кф ×ďĭđāû кääāню

кďî× ññĕîûм¯ûññw кф ¯ûññ ďню

¯ûññ ďĕûĝäĩďĝkм¯ûññ кϑф ¯ûññ күню
```

```
яя ¯ûññщññĕĕ
шîćÑāĭ×û ь¯ûññщёь
кďî× ññĕîûм¯ûññw кф ¯ûññ ďни
    ×ûĩĝĭîĩмкню
    ñĝûûмкф ďщĩäîāāûню
    åďĝмîćĩ îбюîгкщćюîkkн
        кщĩкîлːː ďщĩкîлю
й

¯ûññ ďĕûĝäĩďĝkм¯ûññ кϑф ¯ûññ күни
    äġġûĝĩмкϑщćːːː күщćню
    ¯ûññ кю
    ñĝûûмкф кϑщćню
    åďĝмîćĩ îбюîгкщćю îkkн
        кщĩкîлːː кϑщĩкîлkкүщĩкîлю
    ĝûĩĭĝć кю
й
```

Structures et allocation dynamique

```
яя ¯ÛÑÎщё
ġĬĝĭÑĬ ¯ÛÑÎи
    îćÎ ćю яя ÎÄîĀĀÛ
    ×ĎĭÐĀÛП Îю яя ÎÄÐĀÛÄĭ
йю

кĎî× îćîÎм¯ÛÑÎʷ кню

кĎî× ÑĝÛÛм¯ÛÑÎʷ кф îćÎ ćню

кĎî× ×ÛĬĝĭîÎм¯ÛÑÎʷ кню

кĎî× ĝÛĆĔĀîÎм¯ÛÑÎʷ кф ×ĎĭÐĀÛ кÄĀню

кĎî× ÑĎĔîÛм¯ÛÑÎʷ кф ¯ÛÑÎ Ďню

¯ÛÑÎ ĎĔÛĝÄĬĎĝkм¯ÛÑÎ кϑф ¯ÛÑÎ кϒню
```

```
яя ĆÄîćЩÑĔĔ
Шîć ÑĀĭ×Û ь¯ÛÑÎЩёь

îćÎ ĆÄîćмни
    ¯ÛÑÎ кϑфкϒю
    îćîÎмкϑню
    îćîÎмкϒню

    ÑĝÛÛмкϑф ϑбню
    ĝÛĆĔĀîÎмкϑф фЩѿню

    ÑĎĔîÛмкϒф кϒню

    ¯ÛÑÎ ќϔ̣ кϑ ƙ кϒю

    ×ÛĬĝĭîÎмкϑню
    ×ÛĬĝĭîÎмкϒню
    ×ÛĬĝĭîÎмќϒню
    ĝÛĬĭĝć бю
й
```

Plan de la séance

break

L'instruction **ĐĝÛÄý** permet de sortir d'une boucle.

```
åĎĝмîćĨ î:бЮ îғсЮ îkkни
    ĐĎĎĀ Đ: åмîнЮ
    îåмЦĐн ĐĝÛÄýЮ ЯЯ ġĎĝĨ ×Û ĀÄ ĐĎĭÑĀÛ ġî Đ ÛġĨ åÄĭĽ
й
```

Pour sortir de boucles imbriquées, il faut utiliser des booléens.

```
ĐĎĎĀ ġĨĎĚ: åÄĀġÛЮ
åĎĝмîćĨ î:бЮ îғсЮ îkkни
    åĎĝмîćĨ ú:бЮ úғĆЮ úkkни
        îåмîПú ь ϑббни
            ġĨĎĚ: ĨĝĭÛЮ
            ĐĝÛÄýЮ
        й
    й
    îåмġĨĎĚн ĐĝÛÄýЮ
й
```

```
ĐĎĎĀ æĎ: ĨĝĭÛЮ
åĎĝмîćĨ î:бЮ îғс ww æĎЮ îkkни
    åĎĝмîćĨ ú:бЮ úғĆ ww æĎЮ
     ↪  úkkни
        îåмîПú ь ϑббни
            æĎ: åÄĀġÛЮ
        й
    й
й
```

continue

L'instruction **ÑĎćĨîćĭÛ** permet de passer à l'itération suivante dans une boucle (sans exécuter ce qui se trouve après le **ÑĎćĨîćĭÛ**).

```
îćĨ îˆ̂ˆ̂ϑЮ
ĹëîĀÛмîғ ϑϬϬϬни
    îƙƙЮ
    îåмîкY̆̂̃̂̂ ϑн
        ÑĎćĨîćĭÛЮ
    ÑĎĭĨ ғғ î ғғ Ь ÛġĨ ĚÄîĝЬ ғғ Ûć×ĀЮ
й
```

Plan de la séance