# Mémoire dynamique

30 août 2024 14:29

Une déclaration standard

```
{   //début de portée
    ...
    int x;
    ...
}   //fin de portée
```

dit de réserver un int de mémoire et la libérer en fin de portée automatiquement.

- Le mot-clé

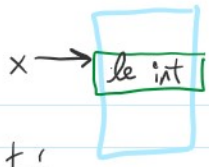
  **new**

  crée une zone mémoire à gérer manuellement.

```
{
    new int;    //créer sizeof(int) mémoire
}   //mémoire non-libérée
```

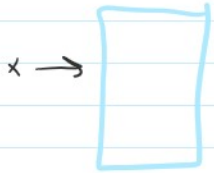
Il faut garder un pointeur (adresse) sur la zone allouée.
Une var. de type pointeur est indiquée avec *

```
int* x = new int;
//...
```




- Pour supprimer la mémoire pointée,

```
delete x;
```




IMPORTANT : chaque new doit être accompagné d'un delete.

Souvent, une fct qui fait un new fait aussi le delete, ou
une classe avec un new fait aussi le

le delete, ou
une classe avec un new fait aussi le delete.

contre-exemple:

```
Point* create_pt() {
    Point* p = new Point();
    return p;
}

Point* pt = create_pt();
```

non

- Opérateur ->

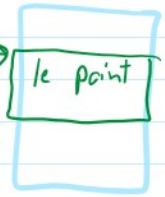
```
Point* p = new Point();
p.set_coord(10, 10);
delete p;
```

ne compile pas

. : objet

p pas un objet

p = pointeur




-> retourne l'objet pointé par un pointeur
pour accéder à un attribut/fct de classe.

```
Point* p = new Point();
p->set_coord(10, 10);
delete p;
```

- Dans une classe,

this

est un pointeur vers l'objet courant.

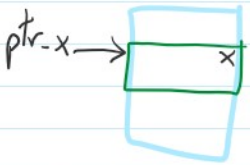
```
void Point::set_coord(double x, double y){
    this->x = x;
    this->y = y;
}
```

est un pointeur vers l'objet courant.

```
void Point::set_coord(double x, double y){
    this->x = x;
    this->y = y;
}
```

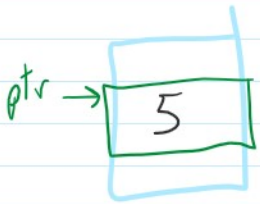
- L'opérateur & retourne en fait un pointeur.

```
int x = 5;
int* ptr_x = &x;
// ne pas faire delete ptr_x;
```




- L'opérateur * sur un pointeur retourne l'objet où le pointeur pointe.

```
int* ptr = new int;
```




```
(*ptr) = 5;
```

(*ptr) : objet pointé

```
cout << ptr << "  " << *ptr; //0x12AB  5
```

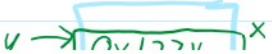
- 
```
Point(const Point& src){
    (*this) = src;
}
```

(*this) : operator =

- Le * effectue un déréférencement.

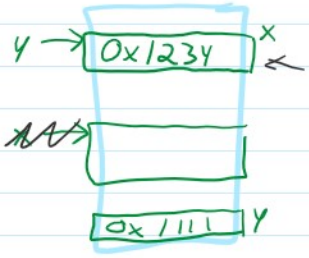
- Pointeur de pointeur

```
int* x = new int;
```



- Pointeur de pointeur

```
int* x = new int;
cout<< &x<< endl;
int** y = &x;
int*** z= &y;
```




- L'opérateur +

```
ptr + k;    //k entier
```

retourne un pointeur qui est k zones mémoires plus loin que ptr.

└ avance de k * sizeof(objet pointé)

- Le mot-clé

```
nullptr
```

qui est un pointeur vers nulle part, vers l'adresse 0.

Utile pour indiquer un pointeur non-initialisé.

```
int* x = nullptr;
```

nullptr est interprété comme 0

```
if (nullptr){
    // ne passe pas
}
```

→

```
if(x){
    // ne passe pas
}
```

Si x avait une adresse autre que 0, ça passerait

```
x = new int;
if (x){
    //passe
}
delete x;
```

---

## Tableaux dynamiques

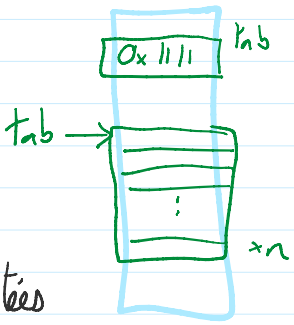
```
int tab[4];      //ok

int n = 10;
int tab[n];      //pas ok

int n;
cin>>n;
int tab[n];      x
```

Pour un tab. dynamique, il faut faire un new pour la création des entrées du tableau.

Syntaxe:

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

Réserve n × sizeof(int) espace contigu à gérer manuellement.



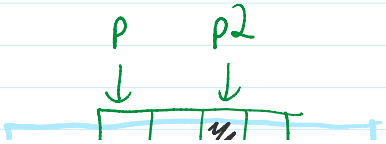

Pour supprimer l'espace des n entrées pointées

```
delete [] tab;
```

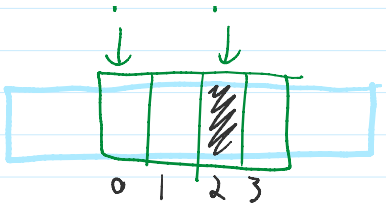
pour libérer tout le tableau. (delete tab; pas suffisant)

- ex: `int n=4;`

```
Point* p = new Point[n];
```



```
Point* p = new Point[n];
p[2].get_x();
Point* p2 = p+2;
(*p2).get_x();
delete [] p;
```



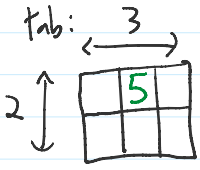

- ptr[i] : va à l'adresse de ptr,
  avance de i entrées,
  retourne l'objet dans cette entrée

- `int tab[4];`
  L'objet tab est un pointeur.
  `int* ptab = tab;`

`f(tab);`

## Tableau à 2 dimensions

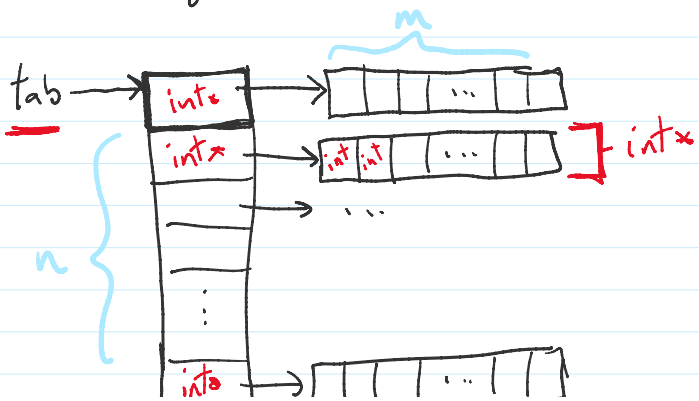
```
int tab[2][3];
tab[0][1] = 5;
```

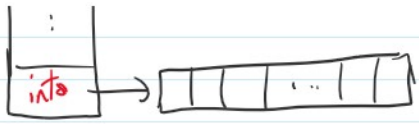



`int tab[n][m]; //non`

- Il faut un tableau de tableaux dynamiques

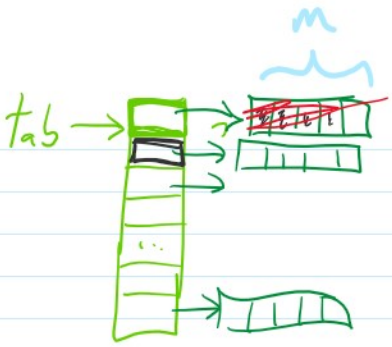
n rangées × m colonnes






```
int** tab;      // tab[i] : un int*
```

Créer un tableau de dim. $n \times m$




```
int** tab = new int*[n];
for (int r=0; r<n; ++r){
    tab[r] = new int[m];
}
// tout init à 0
for (int r=0; r<n; ++r){
    int* tab_r = tab[r];
    for(int c=0; c<m; ++c){
        tab_r[c] = 0;        // tab[r][c] = 0;
    }
}

for (int r=0; r<n; ++r){
    delete [] tab[r];
}
delete [] tab;
```