TP3 → Vector ≠ ~~R~~

stdlib ++ ⇝ "super tableaux"
- ↳ accès à la taille
- ↳ gestion mém. "contrôlée" ↓ utilisateur

t[i] ↓ case i

réalloc (diminuer / agrandir taille)

affectation ....

C ⇝ tableaux t_1, t_2

$t_1 = t_2$

Vector → int : _taille

tableau d'int : _data → int * _data

dynamique

allocation C++ — new int[taille]

où ? Ds constructeurs de Vector.

Vector(int taille)
↳ _data = new int[taille];

règle

objets → new

tableaux → new type[taille]

C++
- constructeur → alloue / initialise
- destructeur → delete // delete[] → libère mém.

delete[] _data;

Constructeur par défaut → val. de base "naturelle"
↳ int / float → 0

↳ Vector v ; → taille ? → tableau ⇒ à utiliser ...

↓

ex : 128 → realloc → augmenter / diminuer ....

---

1. Etant donnée une classe :
class Toto {
public:
Toto(int i) { _data = new float[i] ; _taille = i ; }
private:
float * _data ; int _taille ; } ;

Le destructeur fait : delete _data ; delete _taille (1) 11%

Le destructeur fait : delete _data ; (1) 11%

Le destructeur fait : delete[] _data ; delete _taille ; (1) 11%

Le destructeur fait : delete[] _data ; (6) 67%

_taille → a déjà été créé

1 ?

_taille est un entier donné membre

construit automatiquement avant le constructeur du Vector

détruit auto par le destructeur

---

1. Etant donnée une classe :
class Toto {
public:
Toto(int i) { _data = new float[i] ; _taille = i ; }
private:
float * _data ; int _taille ; } ;

Le destructeur fait : delete _data ; delete _taille (1) 11%

Le destructeur fait : delete _data ; (1) 11%

Le destructeur fait : delete[] _data ; delete _taille ; (1) 11%

Le destructeur fait : delete[] _data ; (6) 67%

✓

_data est 1 ptr → contient l'@ d'un entier

variable _data contient 1@ / ptr construite auto

tableau construit ds le constructeur, par le prog = new ; à détruire manuellement

_data : tableau ↳ delete []

static ⟶ attaché à la classe et non à l'objet

- variables
- méthodes

```
class Toto {
private :
    int _data1;          // 1 par objet
    static int _data2;   // 1 par classe
public :
    int get_data1();
    static int get_data2();
};
```

initialisation → constructeur (quand on crée l'obj.)

init. → au début du .cpp ~ déclaration + init

```
int Toto::_data2 = 4;
```

ex ds main :

```
Toto t1, t2;
t1.get_data1();
t2.get_data1();
```

~~t1.get_data2()~~ ∈ classe

```
Toto::get_data2();
```

meth. statique.

Constructeur par copie

```
Vecto(const Vecto& v)
```

↳ alloue _data taille v._taille

↳ copie les données v._data[i] ⟶ _data[i]

↳ init _taille ....

~~_data = v._data~~

Constr. par copie

Vector(const Vector & v)

→ allouer

→ copier data

→ init _taille

Affectation $v_1 = v_2$

v_1 déjà existant

→ libérer _data

si _data ≠ NULL

ou

si taille ≠

Vector& operator= (const Vector &)

quand utilise-t-on la retour du =

$v_1 = (v_2 = v_3)$ ①

retourne 1 ref s/ v_2

---

int (1)

int & (2) operator[](int i)

→ (1) : on retourne 1 val

→ (2) : on retourne 1 ref

v[i] = 2; ~~v[i] == 2;~~

→ get

→ ~~set~~

v[i] = 2; → get

→ set

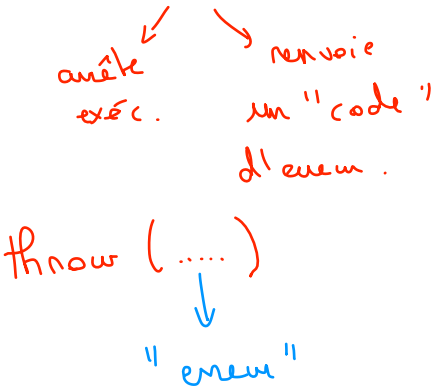
actuellement

cout ... / rien faire

ptr / enum

c++

lever une exception


arrête exéc.
renvoie un "code" d'erreur.
throw (.....)
" erreur "