TP2 Q4

Classe

déclaration .h → class Toto { privé: var / fct ... public: ... };

implémentation des fct .cxx

utilisation → main .... variable de type Toto (classe) → objet

C++

ex: Toto t1 ;

déclaration ⊕ allocation + init

allocation → taille des var. (données membres).

init → on construit l'objet → init des données membres

construction

C ≡ déclaration ⊕ allocation (pas init).

int i ;
float T[10] ;

i ? T ?

mon init

Constructeurs

→ défaut ↔ Dans args : Toto t1 ;

→ avec args : Utilisation : Toto t1(1,2,3) ;

→ par copie : Prototype : Toto (Toto t2) ; (.h)

⚠ efficacité / copies.

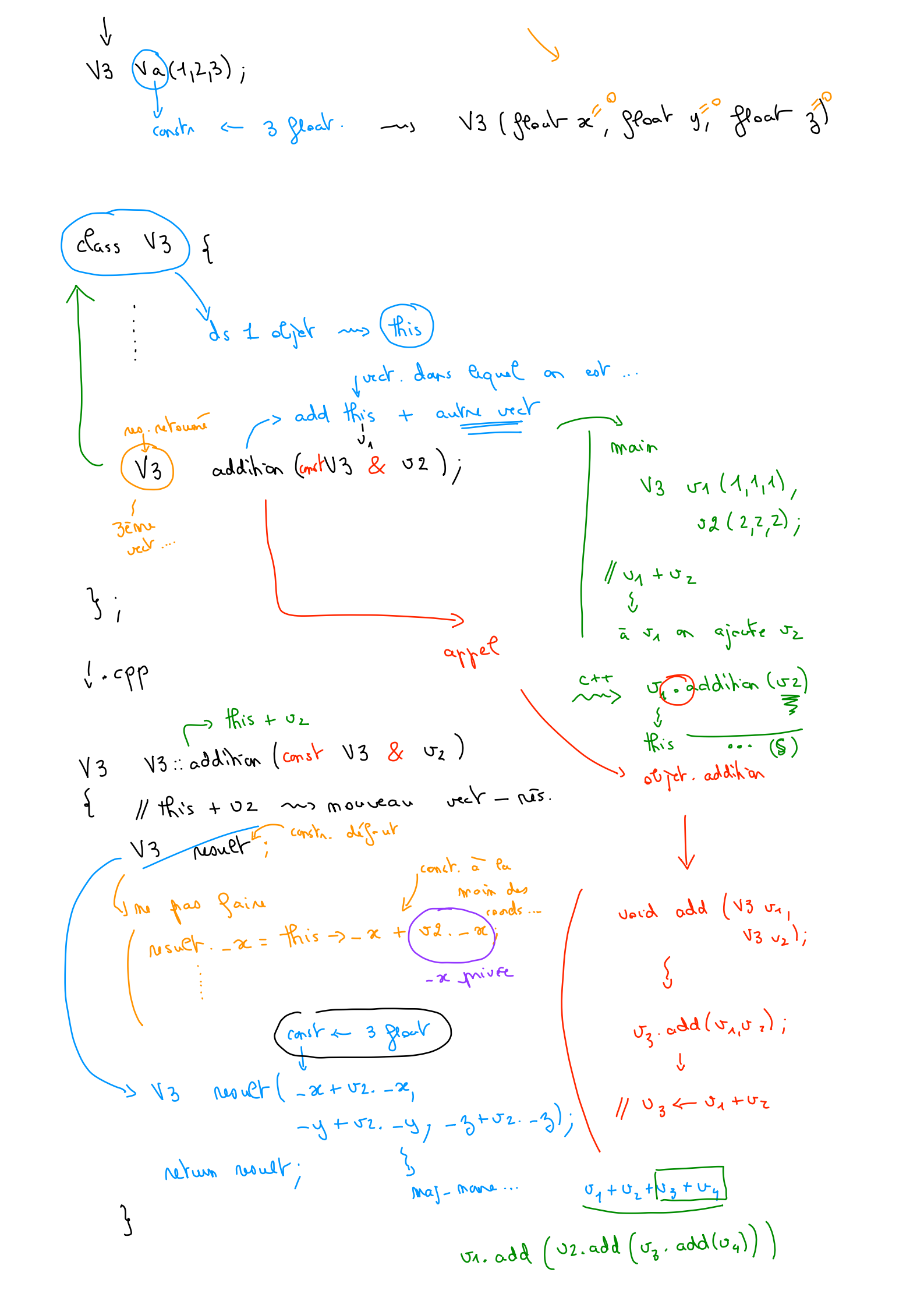
passage par val ⇒ on copie t2 ...

Toto (const Toto & t2) ;

```
V3 Va;
Va._x=1;
Va._y=2;
Va._z=3;
```

→ constr. par défaut (x,y,z ← 0)

privé ....


V3 v1(1,2,3);
constr ← 3 float. ⇝ V3 (float x, float y, float z)
class V3 {
ds 1 objet ⇝ this
vect. dans lequel on est ...
add this + autre vect
v1
réf. retourné
V3
addition (const V3 & v2);
même vect ...
};
.cpp
appel
main
V3 v1(1,1,1),
v2(2,2,2);
// v1 + v2
{
à v1 on ajoute v2
c++
v1.addition(v2)
this ... (§)
objet. addition
this + v2
V3 V3::addition (const V3 & v2)
{ // this + v2 ⇝ nouveau vect – rés.
V3 result; constr. défaut
ne pas faire
result._x = this->_x + v2._x;
constr. à la main des coords ...
-x privée
constr ← 3 float
V3 result(-x + v2._x,
-y + v2._y, -z + v2._z);
maj - mare ...
return result;
}
void add (V3 v1,
V3 v2);
{
v3.add(v1,v2);
// v3 ← v1 + v2
v1 + v2 + v3 + v4
v1.add(v2.add(v3.add(v4)))

main

V3 v3 = v1.addition(v2);

① déclaré → constr. défaut.

② affecté (op. =)

ou

V3 v3(v1.addition(v2));

① constr

V3

constructeur par copie.

⊕ efficace

Constructeurs (.h)

V3(float x, float y, float z);

V3(const V3 & v);

constr. par copie

copie les données de v ds this.

privé → utilisateur

ext. de la classe

ex: main

à l'intérieur de la classe on a accès à tout

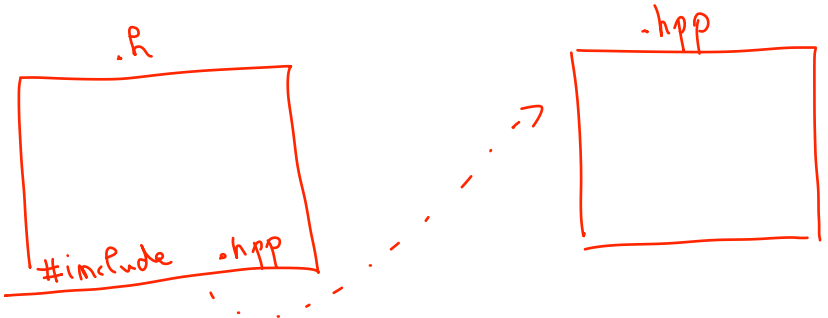
entre 2 obj de même classe ~~privé~~ ...

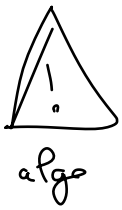
.hpp ← implémentation du .h

code non générique inline → décl + impl. on ne la voit pas ds .h

.hpp ⚠ décl ⇝ .h

code générique – template → impl. ⇝ .hpp





algo

double

pow(x, y) = e^(y.log(x))

x^n = x * ... * x

x^2 = x * x

1 mult

e^(2.log(x))

log → * → exp

```
class V3 {
  :
private:
  _x, ..., _name;
inline void maj_name(void)
  { _name = sqrt(_x*_x ....);
  }
public:
  void set_x(float x) {
    _x = x;
    maj_name();
  }
```

Utilisateur → get_x, set_x

```
v.set_x(3);
```

---

main ⇝

```
V3 v(1,2,3);
V3 *v = new V3(1,2,3);
```

constr. (3 floats).

on crée 1 (1,2,3)

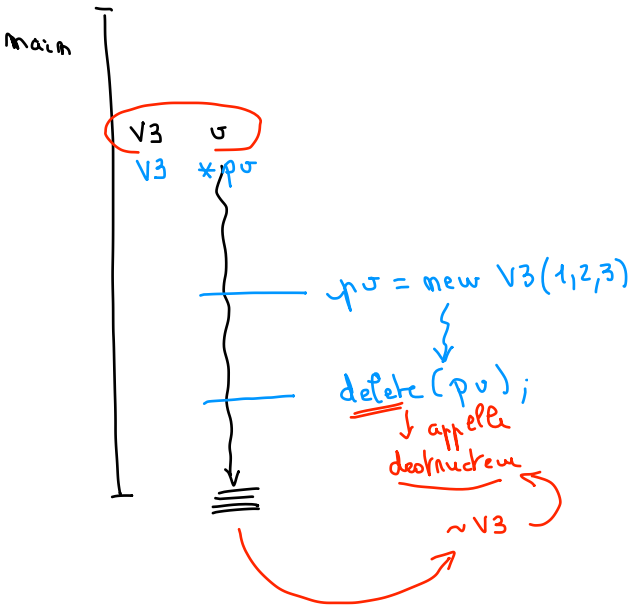
détruit — destructeur appelé

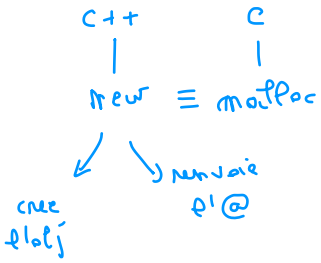
fin V3

fin du prog

delete(v)

destructeur.





"Boulot" du destructeur

classe

données membres

float *T ← tableau

int _i

float _x ....

→ alloc. dynamiques

constructeur

T = new float[10];

destructeur

delete

détruites automatiquement qd l'obj est détruit