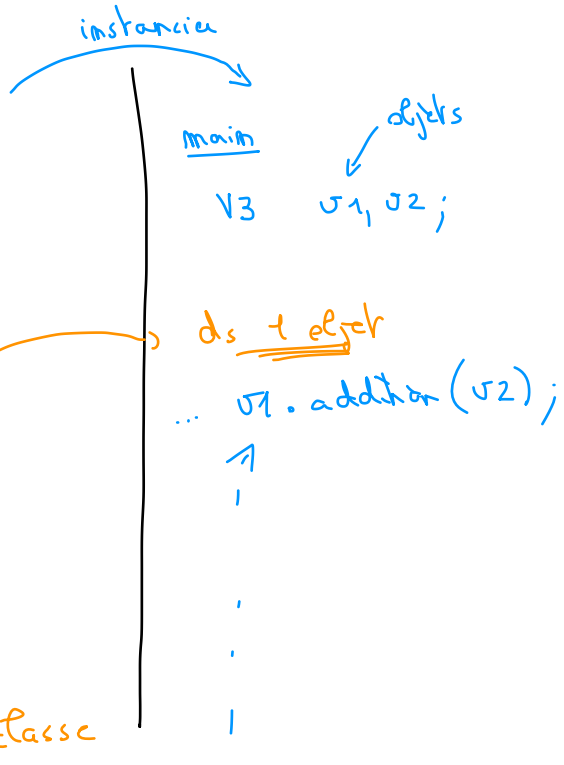


TP2 Q4

```

class V3 {
private:
    float x, y, z, name;
    ...
public:
    V3 addition( ... )
};
    
```



ds + classe.

méthode de la classe

ds + objet

addition (l'obj ds lequel on est) → this

+ V3 argument → 1 argument

addition : this + arg

↓

retourne.

```

class V3 {
    ...
    V3 addition(V3 v)
    }
    
```

(const V3 & v) → passage par ref. constante.

4 float... + mltk.

passage par ~~ref~~ copie...

objets

imp.

```

V3 V3::addition(const V3 & v2)
{
    // retourner "this + v2"
}
    
```

```

V3 tmp (-x + 0.2 * x, -y + 0.2 * y, -z + 0.2 * z);
{ tmp.x = x;
return tmp;
}

```

Diagram showing a function call `V3(...)` returning `tmp`. A blue arrow points from the return statement back to the function call. A red arrow points from the function call to the function definition above.

.h

```

V3(float x, float y, float z)

```

main ↓

```

V3 v1(1, 2, 3);
V3 v2;

```

Diagram: A red arrow points from `v2` to `v1(1, 2, 3)`. A blue arrow points from `v2` to the word "default".

const for copie: ( $\neq$  affectation !)

```

V3(const V3 & v)

```

main ↓

```

V3 v3 = V3(v1);

```

Diagram: A red arrow points from `v3` to `V3(v1)`. A blue arrow points from `v3` to the text "const v3 par default". A blue arrow points from `V3(v1)` to the text "on construit un V3 par copie de v1". A blue arrow points from `V3(v1)` to the text "affectation". A blue arrow points from the text "this ← v" to the `v1` argument.

→ `V3 v3(v1);`

```

v1.add(v2)  ~>  v1 + v2

```

Diagram: A blue arrow points from the `+` operator in `v1 + v2` to the text "operator + / V3".

```

class V3 {
V3 operator + (const V3 & v2)

```

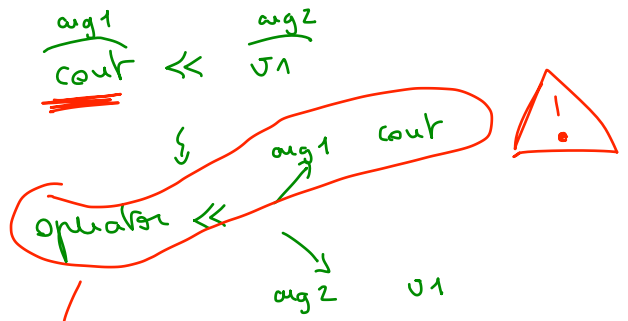
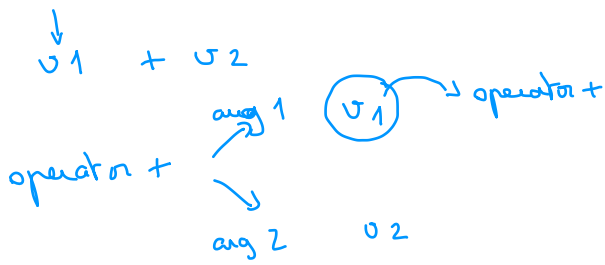
Diagram: A blue arrow points from `const V3 & v2` to `this + v2`. A blue arrow points from `operator +` to `v1.operator+(v2)`. A blue arrow points from `v1 + v2` to `v1.operator+(v2)`. A blue arrow points from `v1 + v2` to the text "this".

```

operator << ~> (cout << v1) << endl;

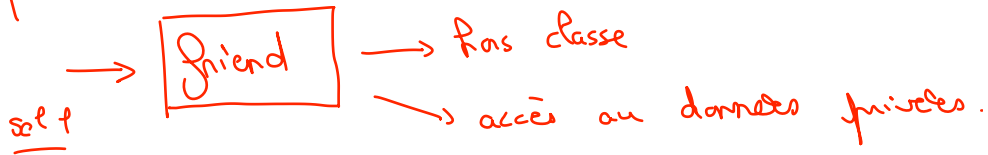
```

Diagram: A red arrow points from `v1` to `cout << v1`. A red arrow points from `cout << v1` to `endl`. A red arrow points from `cout` to `cout`. A red arrow points from `endl` to `endl`. A red arrow points from `endl` to `endl`.



si  $operator \ll$  est ds  $v_3$   
 ↓  
 1er arg (implicite)  
 est this

Pour que  $\ll$  soit hors de la classe:



→ défini hors de la classe  
 ↓  
sol 2

appelle afficher !



class V3

private:

float x, y, z;

public:

inline float get\_x() {  
 return x;}

main

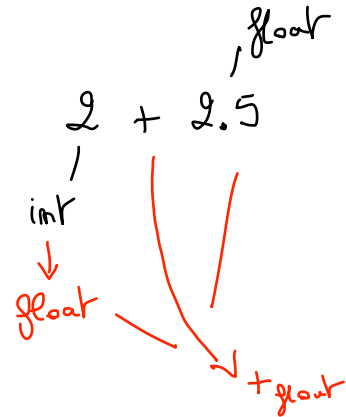
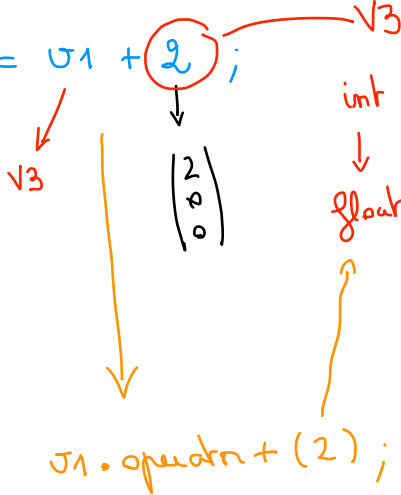
V3 v(1,2,3);

v.get\_x();

```
class V3 {
  V3(float x=0, float y=0, float z=0);
  V3 operator+(const V3 & v);
};
```

```
V3 v1(1,2,3), v2;
```

```
v2 = v1 + 2;
```



~~v2 = 2 + v1~~

→ surcharges symétriques opérateurs  
 → friend  
 → hors classe ←

```
v1 + 1 / 1 + v1 ← OK.
```

→ const sur une méth.

```
V3 addition(const V3 & v)
```

la méth. de modifie pas l'arg.  
 const  
 la méthode ne modifie pas this.

```
V3 {
  get_bidi(const V3 & v)
  v → addition(V3(1,1,1)); ...
}
```

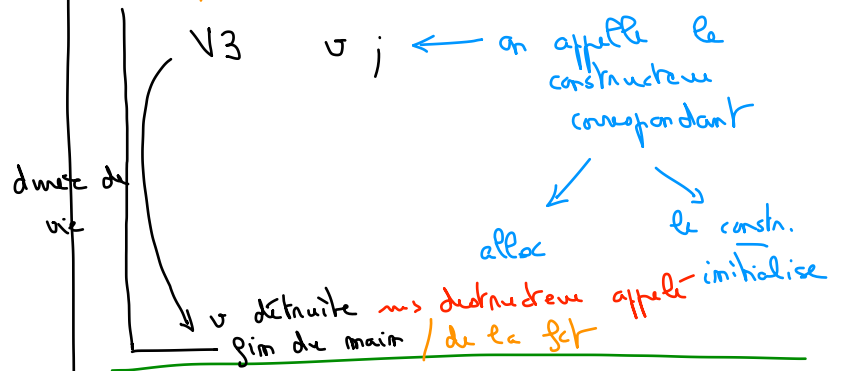
# Destructeur

```
class V3 {
    ...
}
```

destructeur

```
inline ~V3()
{cout << "destructeur"
<<endl; }
```

C++  
main / fonction



```
C
int i;
  déclaration
  ↓
  allocation
  int
```