

Etude de cas 2

Comment calculer l'expression suivante sans produire d'erreur (c.-à-d. sans « Nan », « Not a number ») ?

$$\frac{\sqrt{20 + 7x - x^2} \log\left(\frac{1}{x+5}\right)}{\frac{x}{10} - \sqrt{\log(x^3 - 3x + 7) - \frac{x^2}{5}}}$$

👉 DÉCOMPOSER

Traiter « petit bout par petit bout »

Par exemple :

```
if (x + 5.0 == 0.0) {
  cerr << "Expression invalide pour x=" << x
        << " : division par 0" << endl;
  return 1; // On sort avec un code d'erreur
}
```

Etude de cas

Bien sûr, on suppose qu'au préalable `x` ait été déclaré et ait une valeur, par exemple saisie au clavier :

```
double x(0.0);  
cout << "Entrez une valeur pour x : ";  
cin >> x;
```

On pourrait alors continuer le code par exemple comme suit :

```
double x(0.0);  
cout << "Entrez une valeur pour x : ";  
cin >> x;  
  
if (x + 5.0 == 0.0) {  
    cerr << "Expression invalide pour x=" << x  
        << " : division par 0" << endl;  
    return 1;  
}  
  
if (x + 5.0 < 0.0) {  
    cerr << "Expression invalide pour x=" << x  
        << " : logarithme d'un nombre négatif" << endl;  
    return 1;  
}
```

Etude de cas

Mais ce code présente un **gros défaut** !

```
if (x + 5.0 == 0.0) {  
    cerr << "Expression invalide pour x=" << x  
        << " : division par 0" << endl;  
    return 1;  
}  
  
if (x + 5.0 < 0.0) {  
    cerr << "Expression invalide pour x=" << x  
        << " : logarithme d'un nombre négatif" << endl;  
    return 1;  
}
```

JAMAIS DE « COPIER-COLLER » !

Dans du code, il ne faut **jamais** avoir deux fois la même chose !
(= pour la même raison / de même origine [conceptuelle])

☞ problèmes de maintenance (corrections futures du code)

Etude de cas

Solution : introduire une variable auxiliaire, qui représente justement le fait que ce soit la *même chose* (= la même expression d'origine) :

```
double auxiliaire(x + 5.0);
if (auxiliaire == 0.0) {
    cerr << "Expression invalide pour x=" << x
        << " : division par 0" << endl;
    return 1;
}

if (auxiliaire < 0.0) {
    cerr << "Expression invalide pour x=" << x
        << " : logarithme d'un nombre négatif" << endl;
    return 1;
}
```

Etude de cas

On peut ensuite continuer dans le même esprit, en utilisant si nécessaire une seconde variable :

```
// ...

if (auxiliaire < 0.0) {
    cerr << "Expression invalide pour x=" << x
        << " : logarithme d'un nombre négatif" << endl;
    return 1;
}

double resultat(log(1.0 / auxiliaire));

auxiliaire = 20.0 + 7.0*x - x*x;
if (auxiliaire < 0.0) {
    cerr << "Expression invalide pour x=" << x
        << " : racine d'un nombre négatif" << endl;
    return 1;
}

resultat *= sqrt(auxiliaire);

// etc.
```