Etude de cas

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

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

- ☞ **DÉCOMPOSER**
  Traiter « petit bout par petit bout »
- ☞ **VÉRIFIER** toute condition nécessaire **AVANT** utilisation

Par exemple :

```
if (abs(x + 5.0) < 1e-14) {
  cerr << "Expression invalide pour x=" << x
       << " : division par 0" << endl;
  return 1; // On sort avec un code d'erreur
}
```

Remarques (🚧 pour le moment) :

- pour les `double` : `abs(a - b) < petit` plutôt que `a == b`
  ☞ cf leçon I.4 de théorie (à venir)
- `cerr` est comme `cout` mais doit être préféré pour les messages d'erreur
- `return 1` : par convention : non `0` si 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 (abs(x + 5.0) < 1e-14) {
  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;
}
```


Jean-Cédric Chappelier
& Jamila Sam

Etude de cas

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

```
if (abs(x + 5.0) < 1e-14) {
  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 (générale) : introduire une variable auxiliaire,
qui réprésente justement le fait que ce soit la *même chose* (= la même expression *d'origine*) :

```
double auxiliaire(x + 5.0);
if (abs(auxiliaire) < 1e-14) {
  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;
}
```

Note : dans ce cas précis, *particulier*, on pourrait bien sûr regrouper les deux tests dans un seul et même test unique,
mais comprenez bien le propos *général* !
(p.ex. si l'on *tient* à afficher un message d'erreur différent pour chacun des deux cas ci-dessus)

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.
```