Exercice 0 : reprise de l'exemple du cours (exceptions, niveau 0)

> Cet exercice est aussi détaillé à la page 175 de l'ouvrage *C++ par la pratique*.

Le but de cet exercice est de reprendre l'exemple du cours illustrant la gestion des exceptions.

Commençons par « préparer le terrain » en définissant notre programme de base. Il s'agit 10 fois de suite de demander un nombre, de calculer son inverse par une fonction `f` et d'afficher le résultat.

Aucune difficulté donc ici :

```
#include <iostream>
using namespace std;

double f(double x) {
  if (x == 0.0) {
    return 0.0; // n'importe quoi. C'est ici que nous allons
                // introduire les exceptions
  }
  else return 1.0/x;
}

double demander_nombre() {
// version de base
  cout << " nombre ? ";
  double z;
  cin >> z;
  return z;
}

int main () {
  for (int i(1); i <= 10; ++i) {
    double x, y;
    x = demander_nombre();
    cout << "La valeur de f(" << x <<") ";
    y = f(x);
    cout << "est " << y << endl;
  }
  return 0;
}
```

Compilez et vérifiez que le programme fonctionne.

Introduisons maintenant l'exception pour la division par zéro dans `f`.

Cela se fait en 3 étapes :

1. « lancement » de l'exception. Dans cette première version, lançons simplement une `string` contenant le message d'erreur.

   Cela se fait par l'instruction `throw` au niveau où se produit l'erreur/l'exception :

   ```
   #include <iostream>
   #include <string>
   ... // reste du programme inchangé
   double f(double x) {
     if (x == 0.0) {
       throw string("division par 0");
     }
     else return 1.0/x;
   }
   ...
   ```

2. Prévoir la « capture » de l'exception aux endroits appropriés. Ici, à l'endroit de l'appel de `f` dans `main()`.

Cela se fait par l'ajout d'un bloc « `try` » autour de l'appel :

```
... // reste du programme inchangé
  for (int i(1); i <= 10; ++i) {
    double x, y;
    x = demander_nombre();
    cout << "La valeur de f(" << x <<") ";
    try {
      y = f(x);
      cout << "est " << y << endl;
    }
...
```

3. « capture » de l'exception et gestion par le bloc « `catch` » correspondant.

   Ce bloc doit définir le type des exceptions qu'il gère. Cela se fait par la syntaxe `catch(type& nom)`.

   Dans notre cas, on souhaite « attraper » une `string`.

   De plus dans ce cas simple, la gestion de l'exception consiste simplement à afficher le message. Nous avons donc :

```
... // reste du programme inchangé
  for (int i(1); i <= 10; ++i) {
    double x, y;
    x = demander_nombre();
    cout << "La valeur de f(" << x <<") ";
    try {
      y = f(x);
      cout << "est " << y << endl;
    }
    catch(string& erreur) {
      cerr << "Erreur : " << erreur << endl;
    }
...
```

   Cependant si l'on s'arrête là, la phrase « `La valeur de f(0)` » commencée plus haut ne sera pas terminée en cas d'erreur.

   Il serait plus correct de penser à la conclure :

```
... // reste du programme inchangé
    catch(string& erreur) {
      cout << "n'est pas définie." << endl;
      cerr << "Erreur : " << erreur << endl;
    }
```

Nous avons finit ici la première gestion (élémentaire) de l'exception. Compilez et testez votre programme.

Si cela fonctionne, passons maintenant à une gestion plus avancée des exceptions.

Pour l'instant nous ne passons comme exception qu'un message d'erreur, mais on peut « lancer » n'importe quel objet. En particulier, on pourrait « lancer » plus d'information quant à l'erreur :

- un niveau d'erreur (par exemple~: avertissement, mineure, majeure, critique)
- une indication si l'erreur doit ou non arrêter le programme
- un code d'erreur, identifiant par exemple de quelle erreur il s'agit, où elle s'est produite, etc.
- un message d'erreur
- etc.

Choisissons ici pour illustrer le propos de coder les erreurs par la structure suivante :

```
struct Erreur {
  bool arret;
  unsigned int niveau;
  int code;
  string message;
};
```

Notre programme devient donc (en **gras** les parties modifiées) :

```
#include <iostream>
#include <string>
using namespace std;

struct Erreur {
  bool arret;          // on arrête ou non le programme ?
  unsigned int niveau; // gravité de l'erreur, de 0 à ...
  int code;            // code unique identifiant les erreurs
  string message;      // message d'erreur
};


double f(double x) {
  if (x == 0.0) {
    Erreur err;
    err.arret = false; // par exemple : on ne s'arrête pas sur cette erreur
    err.niveau = 10;  // par exemple
    err.code  = 1;    // code arbitraire identifiant l'erreur
    err.message = "division par 0"; // notre message
    throw err;
  }
  else return 1.0/x;
}

double demander_nombre() {
  ... // code inchangé
}

int main() {
  for (int i(1); i <= 10; ++i) {
    double x, y;
    x = demander_nombre();
    cout << "La valeur de f(" << x <<") ";
    try {
      y = f(x);
      cout << "est " << y << endl;
    }
    catch(Erreur& erreur) {
      if (erreur.code == 1) cout << "n'est pas définie." << endl;
      if (erreur.niveau < 5) cerr << "Avertissement : ";
      else cerr << "Erreur : ";
      cerr << erreur.message << endl;
      if (erreur.arret)
 return erreur.code; // sort du main, donc quitte le programme !
    }
  }
  return 0;
}
```

Compilez et testez ce nouveau code.

Pour finir, introduisons la gestion d'une vraie exception, c'est-à-dire d'un cas singulier mais qui n'est pas une erreur : si l'utilisateur saisit '`q`' au lieu d'un nombre, le programme s'arrête.

Cette exception est « lancée » par la fonction `demander_nombre` et doit être « capturée » dans le `main()`.

Nous déciderons de ne lancer que la touche saisie par l'utilisateur (donc un `char`) :

```
double demander_nombre() {
  cout << " nombre ? [ou q pour sortir] ";
  double z;
  cin >> z;
  if (cin.fail()) { // ce n'est pas un nombre
    cin.clear(); // remet le flot cin dans un état normal
    char lu;
    cin >> lu; // essaye de lire un char
```

```
    if (cin.fail() || (lu != 'q')) {
      // ici c'est une vraie erreur que nous « lançons »
      Erreur err = { true, 2, 2, "saisie erronée" };
      throw err;
    }
    else {
      // ici c'est une vraie exception.
      // on « lance » 'q'
      throw lu;
    }
  }
  return z;
}
```

Le bloc `try` doit maintenant englober aussi l'appel à la fonction `demander_nombre` :

```
int main () {
  for (int i(1); i <= 10; ++i) {
    try {
      double x, y;
      x = demander_nombre();
      cout << "La valeur de f(" << x <<") ";
      y = f(x);
      cout << "est " << y << endl;
    }
...
```

Il nous faut de plus ajouter la gestion de notre vraie exception (la nouvelle erreur est parfaitement gérée par notre ancien bloc `catch`)

```
    catch(char x) {
      if (x == 'q') {
        cout << "bye bye" << endl;
        return 0; // quitte le main()
      }
    }
```

Compilez et testez cette dernière version du programme.

Programme complet :

```
#include <iostream>
#include <string>
using namespace std;

struct Erreur {
  bool arret;          // on arrête ou non le programme ?
  unsigned int niveau; // gravité de l'erreur, de 0 à ...
  int code;            // code unique identifiant les erreurs
  string message;      // message d'erreur
};

double f(double x) {
  if (x == 0.0) {
    Erreur err;
    err.arret = false; // par exemple : on ne s'arrête pas sur cette erreur
    err.niveau = 10;  // par exemple
    err.code  = 1;    // code arbitraire identifiant l'erreur
    err.message = "division par 0"; // notre message
    throw err;
  }
  else return 1.0/x;
}

double demander_nombre() {
  cout << " nombre ? [ou q pour sortir] ";
  double z;
  cin >> z;
  if (cin.fail()) { // ce n'est pas un nombre
    cin.clear(); // remet le flot cin dans un état normal
```

```
    char lu;
    cin >> lu; // essaye de lire un char
    if (cin.fail() || (lu != 'q')) {
      // ici c'est une vraie erreur que nous « lançons »
      Erreur err = { true, 2, 2, "saisie erronée" };
      throw err;
    }
    else {
      // ici c'est une vraie exception (au sens usuel).
      // on « lance » 'q'
      throw lu;
    }
  }
  return z;
}

int main() {
  for (int i(1); i <= 10; ++i) {
    try {
      double x, y;
      x = demander_nombre();
      cout << "La valeur de f(" << x <<") ";
      y = f(x);
      cout << "est " << y << endl;
    }
    catch(Erreur& erreur) {
      if (erreur.code == 1) cout << "n'est pas définie." << endl;
      if (erreur.niveau < 5) cerr << "Avertissement : ";
      else cerr << "Erreur : ";
      cerr << erreur.message << endl;
      if (erreur.arret)
 return erreur.code; // sort du main, donc quitte le programme !
    }
   catch(char x) {
      if (x == 'q') {
        cout << "bye bye" << endl;
        return 0; // quitte le main()
      }
    }
  }
}
```

Dernière mise à jour le 8 septembre 2021
Last modified: Wed Sep 8, 2021