TP POO C++ : Les exceptions

TP POO C++ : Les exceptions

Les TP d'acquisition des fondamentaux visent à construire un socle de connaissances de base, à appréhender un concept, des notions et des modèles qui sont fondamentaux. Ce sont des étapes indispensables pour aborder d'autres apprentissages. Les TP sont conduits de manière fortement guidée pour vous placer le plus souvent dans une situation de découverte et d'apprentissage.

Les objectifs de ce TP sont de découvrir la gestion des exceptions en C++.

Travail demandé

Exception de type entière

La valeur entière de l'exception est ici « vue » comme un **code d'erreur**.

Dans la fonction `Exo_1()`, initialiser une variable entière non nulle `numerateur`, et lire au clavier le dénominateur `denominateur`, lui aussi entier. Tester le dénominateur :

- s'il est non nul, afficher le résultat de la division entière de `numerateur` par `denominateur` (opérateur `/`).
- s'il est nul, lever (`throw`) une exception **entière** (la valeur du `denominateur` par exemple).

Question 1. Dans le fichier source `exo_1a.cpp`, sécuriser l'appel à la fonction `Exo_1()` en l'incluant dans une instruction `try-catch`. Dans le traitement de l'exception capturée (`catch`), afficher la valeur de l'exception dans le flux d'erreur (`cerr`).

Exception de type chaîne de caractères

La chaîne de caractère de l'exception sera donc « vue » ici comme un **message d'erreur**.

Question 2. Dans le fichier source `exo_1aa.cpp`, modifier le programme précédent pour lever une exception "Division par zero" de type `string`. Dans le traitement correspondant, afficher le message associée à l'exception levée.

Relancer des exceptions

L'exception doit être tout d'abord « gérée » dans un bloc `try-catch` au sein de la fonction qui la déclenche puis relancer (`throw`) pour informer celui qui l'a appelée.

Question 3. Dans le fichier source `exo_1b.cpp`, modifier le programme `exo_1a.cpp` précédent pour inclure un bloc `try-catch` dans la fonction fonction `Exo_1()`. Dans le traitement d'exception (`catch`), on relancera l'exception (`throw`) après l'affichage de l'exception. Modifier le corps de la fonction `main()` pour permettre la capture de toute exception quelle qu'elle soit. Dans le traitement correspondant, l'exception ne pouvant être identifiée, afficher simplement le message : Exception interceptée !

Exception de type `range_error`

Il existe 9 classes d'exceptions « prêtes à l'emploi » (`runtime_error`, `bad_alloc`, `out_of_range`, `range_error`, ...) qui héritent de la classe `exception` et qui possèdent un constructeur prenant en argument une chaîne de caractères qui décrit l'exception. On peut ensuite l'afficher en appelant la méthode `what()`. Ces classes sont déclarées dans le fichier d'en-tête `<stdexcept>`.

Question 4. Dans le fichier source `exo_1c.cpp`, modifier le programme précédent pour lever et traiter une exception `range_error`. Modifier le traitement d'exception pour qu'il relance cette exception après l'affichage. Modifier le corps de la fonction `main()` pour permettre la capture d'une exception `range_error`. Dans le traitement correspondant, afficher le message associée à l'exception `range_error`.

Classe d'exception

La classe `exception` encapsule les propriétés et les méthodes fondamentales de toutes les exceptions et fournit une interface pour les applications gérant les exceptions. Il est souvent préférable de créer son type qui hérite de la classe de base `exception`, déclarée dans l'en-tête `<exception>`. Cette classe possède une fonction membre virtuelle `what()` qu'il convient de redéfinir.

Dans la fonction `Exo_2()`, définir une variable caractère `c`, et, dans une boucle infinie, lire au clavier un caractère. Ne sortir de la boucle, en levant une exception `CEMinus`, que si `c` n'est pas une minuscule (cf. `islower`). Il faudra bien entendu utiliser un bloc `try-catch`, même si le traitement d'exception est vide.

Question 5. Dans le fichier source `exo_2.cpp`, vous devez ajouter une classe `CEMinus` par dérivation publique de `exception`. Capturer cette exception et afficher son contenu (toujours par la fonction héritée `what()`).