Exceptions, enums and user interaction

11-04-2022

Motivation

Exceptions in C++ programs are needed in order to interrupt the normal execution of a program when it reaches some logical state that was not provided for when the program was designed. Usually, an exception is thrown by the developer of a function when, in the context of this function, it cannot be correctly handled. However, the caller can properly respond to this by adding an appropriate exception handler.

Beginning programmers often mistakenly use exceptions to “exchange” information between the called function (the *callee*) and the calling function (the *caller*), indicating the failure of the task implemented by the *callee*. Quite often, such a solution turns out to be unsuccessful due to the “expensiveness” of using this mechanism and the availability of simpler and more efficient mechanisms for reporting failure: for example, returning a `false` boolean value or a *null* pointer from a function.

One typical example of failing to handle exceptions is returning a text message from some function via the text block associated with the exception object (accessible via the `what()` method of the standard `std::exception` class). Later, this message is displayed to the end user as part of the normal flow of interaction with him, e.g., through standard streams. At the same time, such textual explanations should be aimed solely at the developer or the tool that conveys this technical information to developers in the final application — e.g., through the sending of an anonymous message authorized by the user.

To manage the behavior of an application where exceptions are possible, it is often useful to develop a custom exception class in which the specific situation that caused the error is encoded with a custom `enum`. Such exceptions are easy to manage and dispatch their processing, incl. output prepared error messages to the end user.

Goal

The aim of this work is to study the correct use of user-defined exceptions to indicate a violation of the program in controlled manner by developing a user-defined exception class with an enumeration of possible failures (codes).

Tasks

As a program for instrumenting user-defined exceptions, it is proposed to take one of the “large” projects from previous lessons, where exceptional situations are possible that are not dependent on the developer. For example, when performing input/output operations (tasks with the “Titanic”).

To implement standard application-wide exception dispatching, follow these steps.

1. Develop a custom exception class `AppExcept`, choose an appropriate base class (e.g., `std::runtime_error` or `std::exception` classes).
2. Define in it a user-defined data type `enum class ErrorCode` describing a list of possible incorrect situations that the program expects.
3. Create a constructor for `AppExcept` that allows you to raise an exception object with an error code and (optionally) an informative message for the developer (specify the ancestor class).
4. In the `main()` function, add a “top-level” exception dispatcher that handles application exceptions (`AppExcept`), all standardized exceptions (`std::exception`), and all others (`...`).
    1. Handle `AppExcept` exceptions a) by the `switch` block, b) by the associated `map` “error code” → “message test”.

Discuss with your teacher possible issues and implement the tasks.