Doc No   : X3J16/95-0093 WG21/N0693
Date   : 19.05.95
Project : Programmming Language C++
Ref.Doc : X3J16/95-0092 WG21/N0692
Reply to : Manfred Lichtmannegger
           manfred.lichtmannegger@mch.sni.de

Title: Rethrowing pending exceptions

Problem:

multiple exceptions

Discussion:

C++ allows only single exceptions (*). If you use exceptions to signal failures, the problem of multiple failures arises.

A straightforward solution would be to throw one exception and queue all other "exceptions" as pending.

After handling the first exception the first, last or most severe exception should be thrown.

As there are many strategies - e.g. FIFO, LIFO, priority based - instead of a built-in strategy a user defined rethrow handler would be adequate.

(*) nested exception can occur during stack unwinding but inner exception must be caught otherwise terminate will be called.

Solution:

The programmer can set an rethrow_handler. When a caught clause is left normally, i.e. not with a throw or a rethrow, as a last action this rethrow_handler - if set - will be executed.

Classification:

It is a small and simple "extension", cheap in terms of implementation and runtime costs.

It can be simulated by declaring a ThrowPending object in each catch clause. Its destructor rethrows on normal exit (*). Interfering catch clauses from third-party-libraries makes life hard. You have to introduce rethrowing guards in your application.

(*) test with predicate throwing or use a simulating idiom

Relations:

Document X3J16/95-0092 WG21/N0692 proposes the predicate throwing. The predicate throwing is useful without "Rethrowing pending exceptions". "Rethrowing pending exceptions" is useful in connection with the predicate throwing, in connection with throw interception as proposed by Gregory Colvin or in connection with the idiom of firewall destructors (Richard Minner).

```
  Proposal:
  ---------

  1) add to paragraph
  -------------------

  15.3 Handling an exception                                    [except.handle]

  ...

  If a handler is left without a throw the rethrow handler (18.6.3) set by
  set_rethrow is called after executing all destructors for local objects and
  the destructor of the caught exception.

  2) add paragraph
  ----------------

  18.6.4  Rethrow handler
[lib.exception.rethrow.handler]

  18.6.4.1  Type rethrow_handler
[lib.rethrow.handler]

  typedef void (*rethrow_handler)();

  The type of a handler function to be called when a catch clause is left
  without a throw.

  18.6.4.2 set_rethrow
[lib.set.rethrow]

  rethrow_handler set_rethrow(rethrow_handler f);

  Effects: Establishes the function designated by f as the current handler
           function which is called when a catch clause is left without a throw.
  Requires: f shall not be a null pointer.
  Returns: The previous rethrow_handler.
  Notes: Initially the rethrow handler is no_rethrow.

  18.6.4.3 no_rethrow
[lib.no.rethrow]

  void no_rethrow();

  The function no_rethrow is the implementation's default rethrow_handler.
  Effects: no effect

  Comment: no_rethrow is a little bit artificial to fulfill the requirement
           of set_rethrow "f shall not be a null pointer" in analogy to
           set_terminate. Without this requirement, no_rethrow can be set to 0,
           provided that a rethrow handler will be only executed when it is not
0.
```