Access to Exception Object's Type Information

X3J16/96-0052, WG21/N0870
Roland Hartinger
Siemens Nixdorf Informationssysteme AG
Munich, Germany

March 6, 1996

1 Problem

If a thrown exception has not been properly caught by an exception handler in a program, the flow of control continues either in `terminate()` or `unexpected()` handler. One reasonable way to act in such cases is to issue a diagnostic message and terminate the program.

Inside those special handlers, there is no portable way to get access to the exception object or to its type information. The same is valid inside a `catch(...)` handler.

However, for diagnostic reasons, it would be of interest to get at least access to the thrown objects type information. This could improve for example, signing off messages which are issued before terminating the program. This is what users need to obtain, to report and explain the problem to their software vendors.

2 Solution 1: using a run-time function

The most obvious solution would be to provide a utility function. The function prototype could look like this:

```
const type_info& caught_exception_typeid();
```

The function returns the type_info of the exception object most recently caught and not finished. So, the programmer could write:

```
void my_terminate_handler() {
    // programs signing off message
    cerr << "Termination caused by: " <<
            caught_exception_typeid().name() << " exception" << endl;
    abort();
}
```

3 Solution 2: supplied by using `typeid()` operator

Another way to obtain the exception object's `type_info` is to (re-)use the `typeid` operator with a special syntax, namely with an empty argument list, for example:

```
void my_unexpected_handler() {
    if (typeid() == typeid(A))
        throw B;    // can rethrow B which can be catched somewhere
else
        terminate_program_with_diagnostic_message(typeid().name());
}
```

4 Implementation aspects

According to the WP, the exception object must remain alive until `terminate()` or `unexpected()` have been finished. The exception run-time system has still access to the type information of the most recently caught exception object.

Therefore, it is cheap to deliver the still available type information (the *type-id* of the exception that would be thrown by a `throw;`) to the caller regardless of using the run-time function or `typeid()` operator.

Using the `typeid()` solution is of more elegance than the function solution and the type information access belongs to it.

5 Recommendation

I recommend to incorporate Solution 2 into the Standard, since it is a very small change in existing language and WP. Furthermore, I have not found any existing implementation which currently has `typeid()` in use. So, there is no compatibility problem with this extension noticed, as far as I can see.

6 Changes to the Working Paper

5.2 Postfix expression [expr.post].

Change the grammar for:

postfix-expression :
  ...
  `typeid` (*expression*)

to

postfix-expression :
  ...
  `typeid` ($expression_{opt}$)

5.2.7 Type identification [expr.typeid].

Add the following paragraph to the end of the section:
If `typeid` *expression* is omitted, `typeid()` will return the *type-id* of the most recently caught but not finished exception (15.5.4). If there is no such exception, `typeid()` throws the `bad_typeid` exception (18.5.3).

Add the following footnote to 5.2.7 :

Footnote: This is the typeid of the exception that would (re-)thrown by `throw;`.

15.5.4 Type information of not finished exceptions .

The most recently caught but not finished exception object's type information can be obtained inside an active `catch(...)`, `terminate()` or `unexpected()` handler.

In such a context, `typeid()` returns a reference to a type_info object that represents the *type-id* of that most recently caught but not finished exception object, otherwise `typeid()` throws the `bad_typeid` exception.

18.5.3 Class `bad_typeid` .

Add to the sentence in paragraph 1 the following:
"... to report a null pointer in a *typeid* expression or thrown by `typeid()` if there is no exception object currently caught and not finished (5.2.7)."