Compromise Proposal for `class exception`

Jerry Schwarz
Greg Colvin
Steve Clamage
Nathan Myers

x3j16-95-0065R1/ WG21 N0665R1

Yesterday, when the library WG discussed Greg Colvin's latest proposal (94-0215/N0602) for class exception we were closer to a consensus that at any recent time. So last night we (Jerry Schwarz, Greg Colvin, Steve Clamage and Nathan Myers) met to see if we could create a proposal that we could all support. We reached agreement on the proposal presented here. This paper provides the rationale for this decision, the various points of view of the participants and precis of other issues we discussed. None of us was completely satisfied with the result but we each felt that our concerns were well enough addressed that we could support this proposal. One factor that heavily influenced our discussion was that we should accept the current WP as a guideline rather than starting over from scratch. Specifically this meant that when we were divided over an issue, but our concerns were not at the "over my dead body" level, we agreed to be guided by the working paper.

The first concern was coupling of `exception's` to `string`. This is a concern that has been expressed repeatedly in discussions of the full committee, and was addressed by Mike Vilot's paper 94-0179/N0566 R1. Although this is a general concern relating to classes derived from exception, the most critical issue is with regard to the `exception`'s that might be thrown by the language support functions. In the library WG some discussion suggested that this coupling could be decreased by changing the type of `exception::what` to take a reference argument. Although this did seem to help, there were still some doubts about it's effectiveness. Instead we propose a change that guarantees the decoupling. Namely change the return type of `exception::what` to const char*.

The second concern was the desire for user code to easily construct, throw and catch exceptions containing dynamically determined messages.

We concluded that this concern could be addressed provided the classes that users would derive from had a constructor that takes a string. Since is already true in the working paper, no changes are needed to address this concern. However, since retaining these in the language support exception classes would conflict with the decoupling concern, we decided to propose eliminating these constructors from `bad_alloc`, `bad_cast`, and `bad_typeid` (also presumably from `XUNEXPECTED` when that class is defined). We also propose moving these classes in the hierarchy to be directly derived from exception. Because of the change to `exception::what` this achieves a similar effect to Mike Vilot's proposal, but retains the advantage of the current WP that all exceptions thrown by the library are derived from exception.

Greg Colvin has a strong desire that the standard should guarantee that certain operations constructing and copying exceptions could be performed without themselves throwing exceptions. Some members of the committee have expressed reservations that this cannot be implemented in any class that contains a string. So we propose only that the classes that don't require a string have throw clauses added. Specifically we propose that we explicitly declare the default constructor, the copy constructor and the assignment operator in exception, `bad_alloc`, `bad_cast`, and `bad_typeid` with a `throw()`.

It should be noted that the copy constructor and assignment of exception might be sliced. That is, the destination might end up with a default message rather than the message it was constructed with. If an implementation can arrange for the message to be copied within the constraint that no exception should be thrown, it should be allowed to do so.

Another concern was voiced by Nathan, who disliked the fact that messages where limited to NTBS (as opposed to a more general character type such as `wchar_t`). Although this proposal does not address this concern, it is certainly no worse than the WP in this regard.

We also discussed the “mixin” approach to the exception hierarchy that has been advocated on the reflector by John Max Skaller. Although this approach has certain appealing aspects there was no concrete proposal before us. We discussed some possibilities but did not reach any firm conclusion. In particular we discussed the desirability of making all derivation in the exception hierarchy virtual. We concluded that this seems reasonable, but there are some potential “gotcha’s” that worried us and we are not prepared to propose anything in this area.

To summarize, the main changes proposed are:

```
virtual const char* exception::what()
```

Language support exception classes derived directly from exception

Prohibiting exception from being thrown for constructing or copying exception (i.e. base class) or language support exception classes.

The specific proposal for changes to the WP

18.4.2.1 [lib.bad.alloc]

Change the class declaration to

```
class bad_alloc : public exception {
public:
    bad_alloc(const bad_alloc &) throw();
    bad_alloc& operator=(const bad_alloc&) throw ();
    bad_alloc() throw() ;
    virtual ~bad_alloc() throw();
    virtual const char* what() const throw();
};
```

[The changes here were to derive directly from exception, add a declaration of the copy constructor and assignment, add throw clauses and to change the return type of what]

Add sections to describe the newly declared constructor and assignment.

18.4.2.1.? [?]

```
bad_alloc(const bad_alloc&) throw();
```

Constructs a bad_alloc. The result of calling `what` on the newly constructed object is implementation defined.

18.4.2.1.? [?]

```
bad_alloc& operator=(const bad_alloc&) throw();
```

The effect of this operation on the return value of `this->what()` is implementation defined.

18.4.2.1.? [?]

~bad_alloc() throw();

Destroys the object.

18.4.2.1.2 [lib.cons.bad.alloc]

Change the description to be

Effects: Constructs an object of class `bad_alloc`.

8.4.2.1.2 [lib.bad.alloc::what]

Change the declaration to

```
virtual const char* what() const ;
```

18.5.2.1 [lib.bad.cast, ...], 18.5.2.2 [lib.bad.typeid,...], 18.6.2.1.

These sections are changed to be copies of 18.4.2 except that a) there is a global substitutions of `bad_cast`, `bad_typeid` and `XUNEXPECTED` for bad_alloc and b) the sentences describing the purpose of the classes are

retained. Specifically in 18.5.2.1: “The class bad_cast defines ....”
and in 18.5.2.2: “The class bad_typeid defines the type ...”

19.1.1 [lib.exception]

The class is modified to

```
class exception {
public:
    exception() throw() ;
    exception(const exception&) throw
    exception& operator=(const exception&) throw() ;
    virtual ~exception() ;
    virtual const char* what() const;
};
```

`throw()` has been added, an explicit assignment has been added, the return type of `what` has been changed, the default constructor has been moved from protected to public, the exposition only members have been removed.]

19.1.1.1 [lib.exception.cons]

Delete the constructor `exception(const string& what_arg)`.

19.1.1.2 [lib.exception.des]

Effects: Destroys an object of class exception. Does not throw any exceptions.

I don’t think we can use a throw clause here because of the way destructors interact with derived classes. That is, I think a `throw()` clause would imply that classes derived from exception don’t throw exceptions and we have explicitly decided not to impose that requirement.

19.1.1.3 [lib.exception::what]

```
virtual const char* what() const ;
```

Returns: an implementation define NTBS.

19.1.1.? [###]

exception& operator=(const exception&) ;

Returns: *this. The result of calling what after an assignment are implementation defined.

19.1.2 [lib.logic.error]

The declaration becomes

```
class logic_error : public exception {
public:
    logic_error(const string& what_arg);
    virtual ~logic_error();
    virtual const char* what() ;
};
```

The declaration of `what` that was exposition only in the current WP is explicit in the proposal.

19.1.2.# [###]

```
const char* what()  ;
```

Returns what_arg.data() where what_arg is the argument used to construct the object.

19.1.3, 19.1.4, 19.1.5, 19.1.6, 19.1.7, 19.1.8, 19.1.9,. 27.4.3.1

Similar changes are made to logic_error, domain_error, invalid_argument, length_error, out_of_range, runtime_error, overflow_error and `ios::failure` respectively. Namely the `what` member is declared explicitly and a definition is provided. No change is made to the inheritance hierarchy. No change is proposed to the inheritance hierarchy of these classes.

Although this is a lot of editorial work the functionality of these classes remains the same as in the current WP. You construct them with a `string` (they have no default constructor) and access their message with `what`.