Deprecating function

Abstract
This paper proposes deprecating function (and associated entities) as it has unresolvable API design issues and has recently been superseded by copyable_function.

Revisions
R0: Initial version

Motivation
C++11 added function, a type-erased function wrapper that can represent any copyable callable matching a given function signature. Since its introduction, there have been identified several issues with its design (see [N4159]) – including the famous constness-bug:

```cpp
// the constness bug of std::function:
// consider:
auto lambda = [&]() mutable { ... };
l();   // ✓
const auto & r(lambda);
r();   // ✓ //lambda::operator() is mutable => can't be called via const &!

// but:
function<void(void)> func(lambda);
func();  // ✓
const auto & cref(func);
cref();  // ✓ //func::operator() is const => can invoke mutable lambda through const &!
```

As function was incompatible (and could not be made compatible) with non-copyable functors, [P0288] introduced move_only_function. The design of which not only drops the copyable requirement but also fixes bugs present in function, removes RTTI dependence and adds support for const-, noexcept- and ref-qualifiers.

Semi-concurrently [P0792] introduced function_ref as a non-owning reference to a functor. Like move_only_function it does not depend on RTTI and supports const- and noexcept-qualifiers².

After move_only_function was approved for C++23 and with function_ref targeting C++26, there were serious inconsistencies between the polymorphic function wrappers of the standard library. Whilst some of the new features³ could have been back-ported to function, it was impossible to reach feature parity without an API break. To improve the situation, [P2548] introduced copyable_function (design consistent with move_only_function) as a replacement of function.

1 RISC Software GmbH, Softwarepark 32a, 4232 Hagenberg, Austria, michael.hava@risc-software.at
2 There is no support for ref-qualifiers as function_ref itself is a reference type.
3 Primarily noexcept- and ref-qualifier support.
With the adoption of copyable_function for C++26 there remains little reason to keep function in the blessed part of the standard library. Furthermore it has been pointed out multiple times (both by [P3023] and externally) that the current state of polymorphic function wrappers in the standard library is complicated - growing from one to four distinct classes within two standard cycles. Deprecating function would lead to a more unified standard library design.

Why now? (Isn’t it too soon?)
Going forward our message to users should be clear: “Avoid function for new code! Instead use the appropriate ‘modern’ polymorphic function wrappers.” Deprecating function in the same standard cycle as the introduction of copyable_function will reduce confusion. Such timing is not entirely novel, it happened before in C++11 with the introduction of unique_ptr and the deprecation of auto_ptr.

Impact on the Standard
Several classes are moved to Annex D without a change in functionality.

Proposed Wording
Wording for the deprecation of function and ancillary entities will be provided in a future revision.

Acknowledgements
Thanks to RISC Software GmbH for supporting this work. Thanks to Zhihao Yuan for providing feedback on an initial draft. Thanks to Peter Kulczycki for proof reading R0.