J16/97-0053R1
WG21/N1091R1
July 16, 1997
J. Stephen Adamczyk
jsa@edg.com

Core II WP Changes (London)

USA core 778:

In 13.3.1p4, add at end:

[Note: no actual type is established for the implicit object parameter of a static member function, and no attempt will be made to determine a conversion sequence for that parameter. See _over.match.best_.]

Replace first two sentences of 13.3.3p1 by:

Define ICSi(F) as follows:

- if F is a static member function, ICS1(F) is defined such that ICS1(F) is neither better nor worse that ICS1(G) for any function G, and, symmetrically, ICS1(G) is neither better nor worse than ICS1(F) [Footnote: If a function is a static member function, this definition means that the first argument, the implied object parameter, has no effect in the determination of whether the function is better or worse than any other function.] ; otherwise,
- [...existing first two sentences of the paragraph.]

./" USA public comment 28 5.3.4p13 expr.new
REPLACE 5.3.4p13:

The allocation function shall either return null or a pointer to a block of storage in which space for the object shall have been reserved. [Note: the block of storage is assumed to be appropriately aligned and of the requested size. The address of the created object will not necessarily be the same as that of the block if the object is an array.]

with:

Unless an allocation function is declared with an empty exception-specification (_except.spec_), throw(), it shall indicate failure to allocate storage by throwing a bad_alloc exception (_except_, _lib.bad.alloc_) and it shall return a non-null pointer otherwise.
If the allocation function is declared with an empty exception-specification, throw(), it shall return null to indicate failure to allocate storage and a non-null pointer otherwise.
When it returns null,
initialization shall not be done, the deallocation function shall not be called, and the value of the new-expression shall be null.

When the allocation function returns a value other than null, it shall be a pointer to a
block of storage in which space for the object shall have been reserved. [Note: the block of storage is assumed to be appropriately aligned and of the requested size. The address of the created object will not necessarily be the same as that of the block if the object is an array.]

AND DELETE 5.3.4p16:

The allocation function can indicate failure by throwing a bad_alloc exception (_except_, _lib.bad.alloc_). In this case no initialization is done.

USA core 774:

Add to end of 5.2.9 as a new paragraph:

An rvalue of type "pointer to cv void" can be explicitly converted to a pointer to object type. A value of type pointer to object converted to "pointer to cv void" and back to the original pointer type will have its original value.

./" USA 52_11/2 5.2.11p2 _expr.const.cast_
REPLACE 5.2.11p2

Any expression may be cast to its own type using a const_cast operator.

WITH

[Note: The conditions imposed on the source type and the target type for a well-formed const_cast conversion can often be satisfied simultaneously by a single type. There is no requirement that the source type and target type be different.]

Similarly, REPLACE the last sentence of 5.2.10p2

Any expression may be cast to its own type using a reinterpret_cast operator.

WITH

[Note: The conditions imposed on the source type and the target type for a well-formed reinterpret_cast conversion can often be satisfied simultaneously by a single type. There is no requirement that the source type and target type be different.]

USA 97-0012/N1050:

As proposed.

USA core 683

In 7.2 paragraph 4, replace the last sentence "The type of an enumerator is its enumeration." with the text

Following the closing brace of an enum-specifier, each enumerator has the type of its enumeration. Before the closing brace, the type of each enumerator is the type of its initializing value. If an initializer is specified for an enumerator, the initializing value has the same type as the expression. If no initializer is specified for the first enumerator, the type is implementation dependent. Otherwise the type is the same as the type of the initializing value of the preceding enumerator unless the incremented value is not representable in that type, in which case the type is

an implementation dependent type sufficient to contain the incremented value.

USA public comment 13:

13.3.1.2 paragraph 9:

If the operator is the operator , , the unary operator &, or the operator ->,
and there are no viable functions, then the operator is assumed to be the built-in operator and interpreted according to clause 5.

USA public comment 16:

Add at end of 9.8p1 sentence 3:

... and has the same access to names outside the function as does the enclosing function.

USA public comment 23:

Add to 5.9p2:

[Example:

```
    void *p;
    const int *q;
    int **pi;
    const int *const *pci;
    void ct()
    {
      p <= q;    // Both converted to "const void *" before comparison
      pi <= pci; // Both converted to "const int **" before comparison
    }
]
```

USA public comment 28:

In 18.1p4, change "null-pointer constant" to "null pointer constant".

USA public comment 36, core 794:

Change 5.2.2p9 to "Recursive calls are permitted, except to the function named
main (_basic.start.main_)."

USA core 773:

Add to 4.2p2:

This conversion is considered only when there is an explicit appropriate pointer target type, and not when there is a general need to convert from an lvalue to an rvalue.

Canada 4, core 776:

In 8.3.6p5, change "The names in the expression ... at the point of

declaration"
to "The names in the expression ... at the point where the default argument expression appears."

Australia 1, Sweden 663

REPLACE the last sentence of 3.9.1p9

An expression of
type void shall be used only as an expression statement (_stmt.expr_), as an operand of a comma expression (_expr.comma_), or as a second or third operand of ?: (_expr.cond_).

WITH

An expression of
type void shall be used only as an expression statement (_stmt.expr_), as an operand of a comma expression (_expr.comma_), as a second or third operand of ?: (_expr.cond_), or as the expression in a return statement (_stmt.return_) for a function with the return type void.

REPLACE all but the first sentence of 6.6.3p2 stmt.return

A return statement with an expression can be used only in functions returning a value; the value of the expression is returned to the caller of the function. If required, the expression is implicitly converted to the return type of the function in which it appears. A return statement can involve the construction and copy of a temporary object (_class.temporary_). Flowing off the end of a function is equivalent to a return with no value; this results in undefined behavior in a value-returning function.

WITH

A return statement with an expression of non-void type can be used only in functions returning a value; the value of the expression is returned to the caller of the function. If required, the expression is implicitly converted to the return type of the function in which it appears. A return statement can involve the construction and copy of a temporary object (_class.temporary_). Flowing off the end of a function is equivalent to a return with no value; this results in undefined behavior in a value-returning function.

A return statement with an expression of type cv void can be used only in functions with a return type of void; the expression is evaluated just before the function returns to its caller.

UK Ed-488:

Replace 5.3.3p6 with: The result is a constant of type size_t. [Note: size_t is defined in <cstddef> (_lib.support.types_).]

France 7:

3.2p4b4: "an expression that is not a null pointer constant, and has type other than void *, is converted ..."

UK Ed-89:

**In 5p3, replace "participate in overload resolution; see _over.match.oper_" with "participate in overload resolution, and as part of that process user-defined
conversions will be considered where necessary to convert class operands to types appropriate for the built-in operator. If a built-in operator is selected, such conversions will be applied to the operands before the operation is considered further according to the rules in this clause; see _over.match.oper_, _over.built_."**

Canada 7, core 752:

Replace 11.2p4 bullet 2 with two bullets:

-- m as a member of N is private, and the reference occurs in a member or friend of class N, or
**-- m as a member of N is protected, and the reference occurs in a member or friend of class N, or in a member or friend of a class P derived from
N, where m as a member of P is private or protected, or**

In 11.5p1 example, change "qualification ignored" to "even though naming class is B".

USA Core 3 1.15:

In 13.3.3p1b3, delete "with the same signature".

USA core 877:

In 13.3p2, change "initialization of a temporary" to "conversion to an lvalue".

Canada 10, core 779, N1084:

Changes as proposed.

USA core 756, 734, 682:

Rewrite of 5.16:

Conditional expressions group right-to-left. The first expression is implicitly converted to bool (_conv_). It is evaluated and if it is true, the result of the conditional expression is the value of the second expression, otherwise that of the third expression. All side effects of the first expression except for destruction of temporaries (_class.temporary_) happen before the second or third expression is evaluated. Only one of the second and third expressions is evaluated.

**If either the second or the third operand has type (possibly cv-qualified) void, then the lvalue-to-rvalue (_conv.lval_), array-to-pointer (_conv.array_),
and function-to-pointer (_conv.func_) standard conversions are performed on the second and third operands, and one of the following shall hold:**

--The second or the third operand (but not both) is a throw-expression (_except.throw_); the result is of the type of the other and is an rvalue.

--Both the second and the third operands have type void; the result is of type void and is an rvalue. [Note: this includes the case where both operands are throw-expressions.]

[begin new text ======>]

Otherwise, if the second and third operand have different types, and either has (possibly cv-qualified) class type, an attempt is made to convert each of those operands to the type of the other. The process for determining whether an operand expression E1 of type T1 can be converted to match an operand expression E2 of type T2 is defined as follows:

- -- If E2 is an lvalue: E1 can be converted to match E2 if E1 can be implicitly converted (_conv_) to the type "reference to T2", subject to the constraint that in the conversion the reference must bind directly (_dcl.init.ref_) to E1.
- -- If E2 is an rvalue, or if the conversion above cannot be done:
  - -- if E1 and E2 have class type, and the underlying class types are the same or inheritance-related: E1 can be converted to match E2 if the class of T2 is the same type as, or a base class of, the class of T1, and the cv-qualification of T2 is the same cv-qualification as, or greater a cv-qualification than, T1. If the conversion is applied, E1 is changed to an rvalue of type T2 that still refers to the original source class object (or the appropriate subobject thereof). [Note: that is, no copy is made.]
  - -- Otherwise: E1 can be converted to match E2 if E1 can be implicitly converted to the type that expression E2 would have if E2 were converted to an rvalue (or the type it has, if E2 is an rvalue).

Using this process, it is determined whether the second operand can be converted to match the third operand, and whether the third operand can be converted to match the second operand. If both can be converted, or one can be converted but the conversion is ambiguous, the program is ill-formed.
If neither can be converted, the operands are left unchanged and further checking is performed as described below. If exactly one conversion is possible, that conversion is applied to the chosen operand and the converted operand is used in place of the original operand for the remainder of this section.

[<======= end new text]

If the second and third operands are lvalues and have the same type, the result is of that type and is an lvalue.

[moved and slightly modified old text======>]

Otherwise, the result is an rvalue. If the second and third operand do not have the same type, and either has (possibly cv-qualified) class type, overload resolution is used to determine the conversions (if any) to be applied to the operands (_over.match.oper_, _over.built_). If the overload resolution fails, the program is ill-formed. Otherwise, the conversions thus determined are applied, and the converted operands are used in place of the original operands for the remainder of this section.

[original old text from here on=======>]

Lvalue-to-rvalue (_conv.lval_), array-to-pointer (_conv.array_), and function-to-pointer (_conv.func_) standard conversions are performed on the second and third operands. After those conversions, one of the following shall hold:

--The second and third operands have the same type; the result is of that

type.
--The second and third operands have arithmetic or enumeration type; the usual arithmetic conversions are performed to bring them to a common type, and the result is of that type.
--The second and third operands have pointer type, or one has pointer type and the other is a null pointer constant; pointer conversions (_conv.ptr_) and qualification conversions (_conv.qual_) are performed to bring them to their composite pointer type (_expr.rel_). The result is of the composite pointer type.
--The second and third operands have pointer to member type, or one has pointer to member type and the other is a null pointer constant; pointer to member conversions (_conv.mem_) and qualification conversions (_conv.qual_) are performed to bring them to a common type, whose cv-qualification shall match the cv-qualification of either the second or the third operand. The result is of the common type.

13.3.1.2 paragraph 7:

... the operands are converted to the types of the corresponding parameters of the selected candidate operation function.

13.6:

Delete paragraphs 25 and 28, moving the note in paragraph 25 to paragraph 26.

Change to restore decay of array rvalues to pointer to element type:

In 4.2p1, change "An lvalue of type..." to "An lvalue or rvalue of type ...".

(end of document)