Core 1 -- Resolution of Comments from National Bodies

References, free store management operators, and additional library facilities.

These extensions to C are summarized in _diff.c_.
The differences between \*C and ISO C are summarized in _diff.iso_.
The extensions to \*C since 1985 are summarized in _diff.c++._

Clauses _lib.library_ through _lib.input.output_ (the "library clauses")

The Standard C library is a subset of the Standard \*C library.

Character set reference
Add a normative reference to ISO/IEC 10646.

---
- The set of "diagnosable semantic rules" consists of all semantic rules in this International Standard except for those rules containing an explicit notation that "no diagnostic is required." or which are described as resulting in "undefined behavior.".
- Every conforming C implementation shall, within its resource limits, accept and correctly execute well-formed C programs, and shall issue at least one diagnostic message when containing an explicit notation that "no diagnostic is required." or which are described as resulting in "undefined behavior.".

Although this International Standard states only requirements on C implementations, those requirements are often easier to understand if they are phrased as requirements on programs, parts of programs, or execution of programs. Such requirements have the following meaning:
Every conforming C implementation shall, within its resource limits, accept and correctly execute.

--- 118,123 -----

Two kinds of implementations are defined:

I hosted

and

--- 118,123 -----

which no diagnostic is required, this International Standard places no requirement on implementations with respect to that program.

P

Two kinds of implementations are defined:

I hosted and

**********************

*** 119,146

and has an implementation-defined set of libraries that includes certain language-support libraries (_lib.compliance_).
Although this International Standard states only requirements on C implementations, those requirements are often easier to understand if they are phrased as requirements on programs, parts of programs, or execution of programs.

Such requirements have the following meaning:

- Whenever this International Standard places a diagnosable requirement on the form of a program (that is, the characters, tokens, syntactic elements, and types that make up the program), and a program does not meet that requirement, the program is ill-formed and the implementation shall issue a diagnostic message when processing that program.

- Whenever this International Standard places a requirement on the execution of a program (that is, the values of data that are used as part of program execution) and the data encountered during execution do not meet that requirement, the behavior of the program is undefined and this International Standard places no requirements at all on the behavior of the program.

In this International Standard, a term is italicized when it is first defined.

In this International Standard, the examples:

--- 128,133 -----
and has an implementation-defined set of libraries that includes certain language-support libraries (_lib.compliance_).

P
In this International Standard, a term is italicized when it is first defined.
In this International Standard, the examples,

**************
*** 155,162
A conforming implementation may have extensions (including additional library functions), provided they do not alter the behavior of any well-formed program.

! One example of such an extension is allowing identifiers to contain characters outside the basic source character set.

Implementations are required to diagnose programs that use such extensions that are ill-formed according to this Standard. Having done so, however, they can compile and execute such programs.

--- 142,148 -----
A conforming implementation may have extensions (including additional library functions), provided they do not alter the behavior of any well-formed program.

! "Japan 23 1 13/7 intro.compliance Extended id chars not extension
Implementations are required to diagnose programs that use such extensions that are ill-formed according to this Standard. Having done so, however, they can compile and execute such programs.

**************
*** 489,495
a most derived object shall have a non-zero size and shall occupy one or more bytes of storage.
Base class sub-objects may have zero size.
! An object of POD type (_basic.types_) shall occupy contiguous bytes of storage.

P
" UK issue 594
N[
a most derived object shall have a non-zero size and shall occupy one or more bytes of storage. Base class sub-objects may have zero size.

Define POD on first use

An object of POD

The acronym POD stands for "plain old data."

type (_basic.types_) shall occupy contiguous bytes of storage.

The observable behavior of the abstract machine is its sequence of reads and writes to volatile data and calls to library I/O functions.

An implementation can offer additional library I/O functions as an extension.
calling a library I/O function, or calling a function that does any of those operations are all

--- 560,569 -----
as \textit{``observable behavior''} as well.

\texttt{.Fe }
\texttt{.P}

\texttt{! Accessing an object designated by a}
\texttt{! ."UK 573  18/6 18/7  intro.execution  Font change for "volatile"
! .CW volatile}
\texttt{! lvalue (_basic.lval_),
modifying an object,
calling a library I/O function, or calling a function that does any of those operations are all
***************
*** 601,606
In other words, function executions do not \texttt{"interleave"} with each other.
\texttt{.Fe }
\texttt{.P}
In the abstract machine, all expressions are evaluated
as specified by the semantics.

--- 601,607 -----
In other words, function executions do not \texttt{"interleave"} with each other.
\texttt{.Fe }
\texttt{+ ."UK 263  18/9  intro.execution  "needed" side effects}
\texttt{.P}
When the processing of the abstract machine is
interrupted by receipt of a signal, the values of objects
***************
*** 602,615
other.
\texttt{.Fe }
\texttt{.P}
- In the abstract machine, all expressions are evaluated
- as specified by the semantics.
- An actual implementation
- need not evaluate part of an expression if it can deduce
- that its value is not used and that no needed side effects
- are produced (including any caused by calling a function or
- accessing a volatile object).
- .P
When the processing of the abstract machine is
interrupted by receipt of a signal, the values of objects
modified after the preceding sequence point are indeterminate

--- 603,608 -----
\texttt{.Fe }
\texttt{."UK 263  18/9  intro.execution  "needed" side effects}
\texttt{.P}
When the processing of the abstract machine is
interrupted by receipt of a signal, the values of objects
\texttt{.\"Germany l  18/_19  intro.execution  Restrictions on signal handlers
***************
*** 612,619
 .P}
When the processing of the abstract machine is
interrupted by receipt of a signal, the values of objects
! modified after the preceding sequence point are indeterminate
! during the execution of the signal handler, and the value of
any object not of
\texttt{.CW "volatile sig_atomic_t"
that is modified by the handler becomes undefined.

When the processing of the abstract machine is
interrupted by receipt of a signal, the values of objects
are unspecified, and the value of
any object not of
that is modified by the handler becomes undefined.

Physical source file characters are mapped, in an implementation-defined
manner, to the source character set
(introducing new-line characters for end-of-line indicators)
if necessary.

Physical source file characters are mapped, in an implementation-defined
manner, to the basic source character set
(introducing new-line characters for end-of-line indicators)
if necessary.

are replaced by corresponding single-character internal representations.
Any source file character not in the basic source character set
is replaced by the universal-character-name that
designates that character.

The process of handling extended
characters is specified in terms of mapping to an encoding that uses only the
basic source character set, and, in the case of character literals and
strings, further mapping to the execution character set.
In practical terms,
however, any internal encoding may be used, so long as an actual extended
character encountered in the input, and the same extended character
expressed in the input as a universal-character-name (i.e. using the
notation), are handled equivalently.

are replaced by corresponding single-character internal representations.
Any source file character not in the basic source character set
is replaced by the universal-character-name that
designates that character.
(An implementation may use any internal encoding, so long as an actual
extended
! character encountered in the source file, and the same extended character
! expressed in the source file as a universal-character-name (i.e. using the
  .CS \euXXXX
notation), are handled equivalently.)
.LI 2
***************
*** 64,71
character encountered in the input, and the same extended character
expressed in the input as a universal-character-name (i.e. using the
  .CS \euXXXX
! notation), are handled equivalently.
! .Fe
  .LI 2
Each instance of
a new-line character and an immediately preceding backslash character
--- 60,66 ------
character encountered in the source file, and the same extended character
expressed in the source file as a universal-character-name (i.e. using the
  .CS \euXXXX
! notation), are handled equivalently.)
.LI 2
Each instance of
a new-line character and an immediately preceding backslash character
***************
*** 121,127
escape sequence,
or universal-character-name
in character literals and string literals
! is converted to a member of the execution character set.
.LI 6
Adjacent ordinary string literal tokens are concatenated.
Adjacent wide string literal tokens are concatenated.
--- 116,124 ------
escape sequence,
or universal-character-name
in character literals and string literals
! is converted to a member of the execution character set
! ." France 1 21/5 2.1 [lex.phases]
! (_lex.ccon_, _lex.string_).
.LI 6
Adjacent ordinary string literal tokens are concatenated.
Adjacent wide string literal tokens are concatenated.
***************
*** 174,180
translator output is collected into a program image which contains
information needed for execution in its execution environment.
.LE
! .H2 "Basic source character set" lex.charset
.P
The
.I basic
--- 171,178 ------
translator output is collected into a program image which contains
information needed for execution in its execution environment.
.LE
! ." Japan 1 21/1 2.2 [lex.charset]
! .H2 "Character sets" lex.charset
.P
The basic consists of 96 characters: the space character, the control characters representing horizontal tab, vertical tab, form feed, and new-line, plus the following 91 graphical characters:
.a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

--- 183,202 ------
consists of 96 characters: the space character, the control characters representing horizontal tab, vertical tab, form feed, and new-line, plus the following 91 graphical characters:
.a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

--- 210,217 ----
The character designated by the universal-character-name \eUNNNNNNN is that character whose encoding in ISO/IEC 10646 is the hexadecimal value \CW NNNNNNNNN; the character designated by the universal-character-name

--- 221,228 ------
The character designated by the universal-character-name \CW \eUNNNNNNNNN is that character whose character short name in ISO/IEC 10646 is \CW NNNNNNNNN; the character designated by the universal-character-name

--- 216,223 ----
the character designated by the universal-character-name \CW \euNNNN is that character whose encoding in ISO/IEC 10646 is the hexadecimal value \CW 0000NNNN.
.H2 "Trigraph sequences" lex.trigraph .P
the character designated by the universal-character-name

If the hexadecimal value for a universal character name

is that character whose encoding in ISO/IEC 10646 is the hexadecimal value

is less than 0x20 or in the range 0x7F-0x9F (inclusive), or if the universal character name designates a character in the basic source character set, then the program is ill-formed.

Before any other processing takes place,

is that character whose character short name in ISO/IEC 10646 is

The basic execution character set and the basic execution wide-character set shall each contain all the members of the basic source character set, plus control characters representing alert, backspace, and carriage return, plus a null character (respectively), whose representation has all zero bits. For each basic execution character set, the values of the members shall be non-negative and distinct from one another.

The execution character set and the execution wide-character set are supersets of the basic execution character set and the basic execution wide-character set, respectively. The values of the members of the execution character sets are implementation-defined, and any additional members are locale-specific.

Before any other processing takes place,
no other trigraph sequence exists. Each that does not begin one of the trigraphs listed above is not changed.

--- 296,303 -----

Each that does not begin one of the trigraphs listed above is not changed.

***************

*** 256,273

Each that does not begin one of the trigraphs listed above is not changed.

--- 301,308 -----

Each that does not begin one of the trigraphs listed above is not changed.

and comments (collectively, \(\text{``white space''}\)), as described below, are ignored except as they serve to separate tokens. Some white space is required to separate otherwise adjacent identifiers, keywords, and literals.
Some white space is required to separate otherwise adjacent identifiers, keywords, and literals.

If it is decimal and has no suffix, it has the first of these types in which its value can be represented:

- int
- long

If it is decimal and has no suffix, it has the first of these types in which its value can be represented:

- int
- unsigned
- long

A decimal integer literal with no suffix
never has type
! .CW unsigned
! .CW int.
! Otherwise, for example, on an implementation where
! .CW unsigned
! .CW int
! values have 16
! bits and
! .CW unsigned
! .CW long
! values have strictly more than 17 bits,
! we would have
! .CW "-30000<0",
! .CW "-50000>0"
! (because
! .CW 50000
! would have type
! .CW unsigned
! .CW int ),
! and
! .CW "-70000<0"
! (because
! .CW 70000
! would have type
! .CW long ).
! .Fe
If it is octal or hexadecimal and has no suffix, it has the first of
these types in which its value can
be represented:
! .CW int,

--- 903,910 -----
! .CW int;
! if the value can not be represented as a
! .CW long
! .CW int,
! the behavior is undefined.
If it is octal or hexadecimal and has no suffix, it has the first of
these types in which its value can
be represented:
! .CW int,

*** ..\basic Tue Jul 15 20:43:32 1997
***************
*** 779,785
namespace N {
    int g(char a)    // overloads N::g(int)
    {
        return k+a;   // k is from unnamed namespace
    }
    .Ce
    .Cb
--- 779,786 -----
namespace N {
    int g(char a)    // overloads N::g(int)
    {
        ."UK 673 335/1 basic.scope.namespace  Fix var name typo
        return l+a;   // l is from unnamed namespace
    }
    .Ce
    .Cb
***************
A name \(``looked up in the context of an expression\)''
is looked up as an unqualified name in the scope where the expression is found.

Because the name of a class is inserted in its class scope (_class_), the name of a class is also considered a member of that class for the purposes of name hiding and lookup.

The set of namespaces is determined entirely by the types of the function arguments.

Typedef names used to specify the types do not contribute to this set. The set of namespaces are determined in the following way:

To be considered.
allows a global name to be referred to even if its identifier has been hidden (_basic.scope.hiding_).

P
A
.I nested-name-specifier
! that names a scalar type, followed by
! .CW :: ,
! followed by
! .I ~type-name
! is a
 .I pseudo-destructor-name
  for a scalar type (_expr.pseudo_).

The

allows a global name to be referred to even if its identifier has been hidden (_basic.scope.hiding_).

P
."USA CD2-core-3, core-665 343/5 basic.lookup.qual Pseudo-destructor lookup
If a
 .I pseudo-destructor-name
 (_expr.pseudo_) contains a
 .I nested-name-specifier,

***************

** 1525,1534
 .I ~type-name
  is a
 .I pseudo-destructor-name
 ! for a scalar type (_expr.pseudo_).
 ! The
 ! .I type-name
 ! is looked up as a type in the scope of the
 .I nested-name-specifier .
 .E[
 .Cb

--- 1530,1540 -----
."USA CD2-core-3, core-665 343/5 basic.lookup.qual Pseudo-destructor lookup
If a
 .I pseudo-destructor-name
 (_expr.pseudo_) contains a
 .I nested-name-specifier ,
 ! the
 ! .I type-name s
 ! are looked up as types in the scope designated by the
 .I nested-name-specifier .
In a
 .I qualified-id

***************

** 1530,1535
 .I type-name
 is looked up as a type in the scope of the
 .I nested-name-specifier .
 .E[
 .Cb

 struct A {
+ In a
+ .I qualified-id
+ of the form:
+ + .Cb
+ \textit{\texttt{\fP::\f2*(op nested-name-specifier\fP ~ \f2class-name}}
+ + .Ce
+ where the
+ + .I nested-name-specifier
+ designates a namespace scope, and in a
+ + .I qualified-id
+ of the form:
+ + .Cb
+ \textit{\texttt{\fP::\f2*(op nested-name-specifier class-name\fP :: ~ \f2class-name}}
+ + .Ce
+ + .I class-name s
+ + are looked up as types in the scope designated by the
+ + .I nested-name-specifier.
+ .E[
+ .E[
+ .Cb
+ struct A {
+ ******************
+ *** 2046,2051
+ If the name is found in both contexts, the
+ .I class-name-or-namespace-name
+ shall refer to the same entity.
+ .N[
+ because the name of a class is inserted in its class scope
+ (_\texttt{\_class_}), the name of a class is also considered a nested
+ --- 2070,2076 -----
+ If the name is found in both contexts, the
+ .I class-name-or-namespace-name
+ shall refer to the same entity.
+ + ."Core issue 869 3.4.5p3
+ .N[
+ the result of looking up the
+ .I class-name-or-namespace-name
+ ******************
+ *** 2047,2057
+ .I class-name-or-namespace-name
+ shall refer to the same entity.
+ .N[
+ - because the name of a class is inserted in its class scope
+ - (_\texttt{\_class_}), the name of a class is also considered a nested
+ - member of that class.
+ - .N] e
+ - .N[
+ the result of looking up the
+ .I class-name-or-namespace-name
+ is not required to be a unique base class of the class type of the object
+ --- 2072,2077 -----
+ shall refer to the same entity.
+ ."Core issue 869 3.4.5p3
+ .N[
+ the result of looking up the
+ .I class-name-or-namespace-name
+ is not required to be a unique base class of the class type of the object
+ ******************
+ *** 2075,2081
+ e.B::a = 0;  // ok, only one A::a in E
+ F f;
The function `main` shall not be called from within a program.

The linkage (\_basic\_link\_) of
- The function `main`
- A program that takes the address of
  - `main`
- A program that declares
  - `main`

A program that declares
- `main`
- to be
  - `main`

- `main`
- `main`
- `main`
- `main`
- `main`

It is implementation-defined whether the dynamic initialization
- `main`
- `main`
- `main`
- `main`
- `main`
of an object of namespace scope with static storage duration is done before the first statement of

--- 2548,2555 -----

.N]
.P
.iix "unspecified order-of evaluation
! ."France 8 362/3 basic.start.init Deferred vs guaranteed init
! It is implementation-defined whether or not the dynamic initialization (_dcl.init_, _class.static_, _class.ctor_, _class.expl.init_) of an object of namespace scope is done before the first statement of

***************
*** 2526,2532
.iix "unspecified order-of evaluation
It is implementation-defined whether the dynamic initialization (_dcl.init_, _class.static_, _class.ctor_, _class.expl.init_) of an object of namespace scope with static storage duration is done before the first statement of

.CW main
or deferred to any point in time after the first statement of

--- 2551,2557 -----

."France 8 362/3 basic.start.init Deferred vs guaranteed init
It is implementation-defined whether or not the dynamic initialization (_dcl.init_, _class.static_, _class.ctor_, _class.expl.init_) of an object of namespace scope is done before the first statement of

.CW main.
If the initialization is deferred to some point in time after

***************
*** 2528,2538
(_dcl.init_, _class.static_, _class.ctor_, _class.expl.init_) of an object of namespace scope with static storage duration is done before the first statement of

! .CW main
! or deferred to any point in time after the first statement of
! .CW main
! but before the first use of a function or object defined in the same translation
! unit.
! .E[
! .Cb

// -- File 1 --

--- 2553,2569 -----
(_dcl.init_, _class.static_, _class.ctor_, _class.expl.init_) of an object of namespace scope is done before the first statement of

! .CW main.
! If the initialization is deferred to some point in time after
! the first statement of
! .CW main.
! it shall occur before the first use of any function or object
! defined in the same translation unit as the object to be initialized./*f
! .Fs
! An object defined in namespace scope having initialization with side-effects must be initialized even if it is not used
! (_basic.stc.static_).
! .Fe
! .E[
// -- File 1 --
***************
*** 2559,2565
    b.Use();
}

! It is implementation-defined whether
    .CW a
    is defined before
    .CW main
--- 2590,2596 -----
    b.Use();
}

! It is implementation-defined whether either
    .CW a
    or
    .CW b
***************
*** 2561,2567
    .Ce
    It is implementation-defined whether
    .CW a
    ! is defined before
    .CW main
    is entered or whether its definition is delayed until
    .CW a
--- 2592,2600 -----
    .Ce
    It is implementation-defined whether either
    .CW a
    ! or
    ! .CW b
    ! is initialized before
    .CW main
    is entered or whether the initializations are delayed until
    .CW a
***************
*** 2563,2569
    .CW a
    is defined before
    .CW main
    ! is entered or whether its definition is delayed until
    .CW a
    is first used in
    .CW main .
--- 2596,2602 -----
    .CW b
    is initialized before
    .CW main
    ! is entered or whether the initializations are delayed until
    .CW a
    is first used in
    .CW main .
***************
*** 2567,2580
    .CW a
    is first used in
    .CW main .
- It is implementation-defined whether
- `.CW b`
- is defined before
- `.CW main`
- is entered or whether its definition is delayed until
- `.CW b`
- is first used in
- `.CW main`
  In particular,
  if
  `.CW a`

--- 2600,2605 -----
  `.CW a`
  is first used in
  `.CW main`
  In particular,
  if
  `.CW a`
  **************
*** 2578,2584
  In particular,
  if
  `.CW a`
  ! is defined before
  `.CW main`
  is entered,
  it is not guaranteed that

--- 2603,2609 -----
  In particular,
  if
  `.CW a`
  ! is initialized before
  `.CW main`
  is entered,
  it is not guaranteed that
  **************
*** 2587,2593
  `.CW a`,
  that is, before
  `.CW A::A`
  ! is called.
  `.E] e`
  `.P`
  If construction or destruction of a non-local static object

--- 2612,2624 -----
  `.CW a`,
  that is, before
  `.CW A::A`
  ! is called. If, however,
  ! `.CW a`
  ! is initialized at some point after the first statement of
  ! `.CW main`,
  ! `.CW b`
  ! will be initialized prior to its use in
  ! `.CW A::A`.
  `.E] e`
  `.P`
  If construction or destruction of a non-local static object
  ********************
*** 2632,2638
  `_lib.support.start.term_`) then following the call to
  `.CW exit`,
any objects with static storage duration initialized prior to the registration
of that function will not be destroyed until the registered function is called
from the termination process and has completed.
For an object with static storage duration constructed after a function is registered with

--- 2663,2669 ------
_lib.support.start.term_) then following the call to
_CW exit,
any objects with static storage duration initialized prior to the registration
of that function shall not be destroyed until the registered function is called
from the termination process and has completed.
For an object with static storage duration constructed after a function is registered with
***************
*** 2641,2646
_CW exit,
the registered function is not called until the execution of the object's destructor has completed.
.P
_ix "[abort]
_ix "program termination

--- 2672,2683 ------
_CW exit,
the registered function is not called until the execution of the object's destructor has completed.
+ .\"Canada 1 363/3 basic.start.term Interleaving atexit
+ If
+ _CW atexit
+ is called during the construction of an object, the complete object to which it belongs shall be destroyed before the
+ registered function is called.
.P
_ix "[abort]
_ix "program termination
***************
*** 2684,2691
_CW auto
are related to storage duration as described below.
.P
! References (_dcl.ref_) might or might not require storage; however, the storage
duration categories apply to references as well.
.H3 "Static storage duration" basic.stc.static
.P
All objects which neither have dynamic storage duration nor are local have

--- 2721,2729 ------
_CW auto
are related to storage duration as described below.
.P
! .\"France 9 38/ basic.life Lifetime of references
! The storage duration categories apply to references as well. The
! lifetime of a reference is its storage duration.
.H3 "Static storage duration" basic.stc.static
.P
All objects which neither have dynamic storage duration nor are local have
The requirements of this paragraph shall be met for each allowable ordering of the subexpressions of a full expression; otherwise the behavior is undefined.

```c
i = v[i++];  // the behavior is undefined
```

The requirements of this paragraph shall be met for each allowable ordering of the subexpressions of a full expression; otherwise the behavior is undefined.

```c
i = v[i++];  // the behavior is unspecified
```

```c
--- 53,59 -----  
```

```c
i = 7, i++, i++;  // 'i' becomes 9
```

```c
i = ++i + 1;   // the behavior is undefined
```

```c
--- 55,61 ------  
```

```c
--- 56,62 ------  
```

```c
--- 58,64 ------  
```

```c
--- 59,65 ------  
```

```c
--- 832,837 ----  
```

The requirements of this paragraph shall be met for each allowable ordering of the subexpressions of a full expression; otherwise the behavior is undefined.

```c
i = v[i++];  // the behavior is undefined
```

```c
--- 53,58 ------  
```

i = v[i++];  // the behavior is unspecified

```c
i = 7, i++, i++;  // 'i' becomes 9
```

```c
i = ++i + 1;   // the behavior is unspecified
```

```c
--- 55,61 ------  
```

```c
--- 56,62 ------  
```

```c
--- 58,64 ------  
```

```c
--- 59,65 ------  
```

```c
--- 832,837 ----  
```

```c
--- 55,61 ------  
```

```c
--- 56,62 ------  
```

```c
--- 58,64 ------  
```

```c
--- 59,65 ------  
```

```c
--- 832,837 ----  
```

```c
--- 55,61 ------  
```

```c
--- 56,62 ------  
```

```c
--- 58,64 ------  
```

```c
--- 59,65 ------  
```

```c
--- 832,837 ----  
```

```c
--- 55,61 ------  
```

```c
--- 56,62 ------  
```

```c
--- 58,64 ------  
```

```c
--- 59,65 ------  
```

```c
--- 832,837 ----  
```
The result is a constant of an implementation-defined type
which is the same type as that which is named
! .CW size_t
 .ix "implementation-defined type of [size_t]
 .ix "implementation-defined [sizeof] expression
in the standard header

The result is a constant with type
! .CW size_t
 .ix "implementation-defined type of [size_t]
 .ix "implementation-defined [sizeof] expression
as defined in the standard header

in the standard header
 .CW <cstdlib> (_lib.support.types_).
 .ix "[<cstdlib>]
 .ix "[size_t]

is the pointer declarator and not the multiplication operator.
 .E] e
 .P
! Parentheses shall not appear in the
is the pointer declarator and not the multiplication operator.

--- 2553,2561 -----
parentheses in a
new-type-id
of a
new-expression
**************

--- 2558,2566 -----
parentheses in a
new-type-id
of a
new-expression

--- 2675,2682 -----
is obtained from the appropriate \f2 allocation function\fP (_basic.stc.dynamic.allocation_).

--- 2676,2676
_alloc allocation function
+ ." USA CD-2 core 1-9 core 753
+ ." allow creating objects of class types into a char array
When the allocation function is called, the first argument shall be the amount of space requested.
If the object being created is not an array, the size requested shall be the

**************
An implementation shall provide default definitions for the global

! size of the object.
! If the object is an array, the size requested may be larger than the size
! of the object.
! For arrays of
! .CW char
! and
! .CW unsigned
! .CW char,
! the difference between the result of the new expression and the address
! returned by the allocation function shall be an integral multiple of the
! most stringent alignment requirement (.basic.types_) of any object type
! whose size is no greater than the size of the array being created.
! .N[ since allocation functions are assumed to return pointers to storage that
! appropriately aligned for objects of any type, this constraint on array
! allocation overhead permits the common idiom of allocating character
! arrays into which objects of other types will later be placed.
! .N] e

An implementation shall provide default definitions for the global

--- 545,550 -----
If a function with external linkage is declared inline in one translation
unit, it shall be declared inline in all translation units in which it appears;
no diagnostic is required.

--- 545,555 -----+
If a function with external linkage is declared inline in one translation
unit, it shall be declared inline in all translation units in which it appears;
no diagnostic is required.
An object declared with a const-qualified type has internal linkage
! unless it is explicitly declared
! or unless it was previously declared to have external linkage.
A variable of const-qualified integral or enumeration type

--- 836,844 ----
types.
.N] e
.P
! ." Canada 3
! An object declared in namespace scope with a const-qualified type has internal
! linkage unless it is explicitly declared
! or unless it was previously declared to have external linkage.
A variable of const-qualified integral or enumeration type

**************
*** 1650,1656
can be used to redefine a
.I namespace-alias
declared in that declarative region
! to refer to the namespace to which it already refers.
.E[
the following declarations are well-formed:
.Cb

--- 1656,1663 ----
can be used to redefine a
.I namespace-alias
declared in that declarative region
! ." Editorial: UK ED-114
! to refer only to the namespace to which it already refers.
.E[
the following declarations are well-formed:
.Cb

**************
*** 1759,1771
since constructors and destructors do not have names, a
.I using-declaration
cannot refer to a constructor or a destructor for a base class.
- A
- .I using-declaration
- can refer to a base class copy-assignment operator; however, this
- copy-assignment operator is never used as the copy-assignment operator
- for the
- derived class that contains the
- .I using-declaration
- (_over.ass_).
.N] e
.P
A

--- 1766,1771 ----
since constructors and destructors do not have names, a
.I using-declaration
cannot refer to a constructor or a destructor for a base class.
.N] e
." USA CD2-core 1-5 core-672
." Using-declarations cannot introduce copy-assignment operators
I using-declaration

--- 1767,1783 -----

.I using-declaration

cannot refer to a constructor or a destructor for a base class.

.N e + ." USA CD2-core 1-5 core-672
+ ." Using-declarations cannot introduce copy-assignment operators
+ If an assignment operator brought from a base class into a derived class
+ has the signature of a copy-assignment operator for the derived class
+ (.class.copy_), the
+ .I using-declaration
+ does not by itself suppress the implicit declaration of the derived class
+ copy-assignment operator;
+ the copy-assignment operator from the base class is hidden or overridden
by
+ the implicitly-declared copy-assignment operator of the derived class, as
+ described below.

.P A
-I using-declaration

***************

--- 2409,2414 -----

.or .CW FORTRAN
(depending on the vintage).

.N e + ." Germany 5
+ The semantics of a language linkage other than \*C or C are
+ implementation-defined.

.P .P

.Ix "implementation-defined linkage-specification

--- 2420,2428 -----

.or .CW FORTRAN
(depending on the vintage).
+ ." Germany 5
+ The semantics of a language linkage other than \*C or C are
+ implementation-defined.

.P .P

.Ix "implementation-defined linkage-specification

***************

--- 2457,2463 -----

.Ce .E]

.Ix "linkage-specification class
! A non-\*C language linkage is ignored for the names of class members and
for the function type of class member function declarators.

.Ce .Cb

--- 2471,2480 -----

.Ce .E]

.Ix "linkage-specification class
! ." Germany 5
! ." Semantics of language linkages other than \*C or C are
A C language linkage is ignored for the names of class members and for the function type of class member function declarators.

--- 2593,2608 -----
extern "C" {
    int i;    // definition
}

--- 2576,2581 -----
extern "C" {
    int i;    // definition
}

--- 2244,2249 -----
is ill-formed.

--- 2248,2258 -----
.I initializer s
in the list than there are members in the aggregate,

--- 2244,2251 -----
is ill-formed.

--- decl Tue Jul 15 20:43:46 1997
decl.new Tue Jul 15 05:53:18 1997
--- USA CD2-core 1-6 core-746
--- a declaration with both extern "C" and a storage class is ill-formed
--- A linkage-specification
directly containing a single declaration shall not specify a storage class.

--- 2244,2251 -----
is ill-formed.

--- Canada 5
default initialization instead of initialization with T() covers array case
If there are fewer
.I initializer s
in the list than there are members in the aggregate,
struct S { int a; char* b; int c; };

--- 2250,2256 ------
.I initializer s
in the list than there are members in the aggregate,
then each member not explicitly initialized
! shall be default-initialized (_dcl.init_).

***************

*** 2482,2487
.N] e
.H3 "Character arrays" dcl.init.string
.P
.A
.CW char
.array (whether plain

--- 2480,2487 ------
.N] e
.H3 "Character arrays" dcl.init.string
.P
+ ."USA CD2-core-7 core-751
+ ."redundant curly braces allowed around a string-literal initializing an
.array.
.A
.CW char
.array (whether plain

***************

*** 2492,2498
.CW unsigned
.CW char )
can be initialized by a
! .I string-literal ;
a
.CW wchar_t
.array can be initialized by a wide

--- 2492,2499 ------
.CW unsigned
.CW char )
can be initialized by a
! .I string-literal
! (optionally enclosed in braces);
a
.CW wchar_t
.array can be initialized by a wide

***************

*** 2496,2502
.a
.CW wchar_t
.array can be initialized by a wide
! .I string-literal
! .ix "character-array initialization
.successive characters of the
! .I string-literal

--- 2497,2504 ------
a
.CW wchar_t
.array can be initialized by a wide
The acronym POD stands for "plain ol\' data." is an aggregate class that has no non-static data members of type pointer to member, non-POD-struct, non-POD-union (or array of such types) or reference,

A union shall not have base classes.
A union shall not be used as a base class.
An object of a class with a non-trivial default constructor (_class.ctor_),
a non-trivial copy constructor (_class.copy_),
a non-trivial destructor (_class.dtor_),
or a non-trivial copy assignment operator (_over.ass_, _class.copy_)
Anonymous unions declared in unnamed namespaces do not have to be declared static.
Anonymous unions declared in a named namespace or in the global namespace shall be declared static.
Anonymous unions declared at block scope shall be declared with any storage class allowed for a block-scope variable, or with

```c
struct C;
void no_opt(C*);

struct C {
  int c;
  C() : c(0) { no_opt(this); }
};

const C cobj;

void no_opt(C* cptr) {
  int i = cobj.c * 100; // value of cobj.c is unspecified
  cptr->c = 1;
  cout << cobj.c * 100 // value of cobj.c is unspecified
       << '\n';
}
```

Temporary objects have special semantics when used during construction; see _class.base.init_ and _class.cdtor_.

--- 248,283 -----
some language constructs have special semantics when used during construction;
see _class.base.init_ and _class.cdtor_.

--- 248,283 ---
`no_opt` is a function that may access a `C` object through a lvalue that is not obtained, directly or indirectly, from the constructor's `this` pointer, the value of the object or subobject thus obtained is unspecified.
class C {
    // ...

--- 424,431 ------
    .CW obj2
    is destroyed.
    .E[
    + ."Canada 8
    + ."Fix operator+ to return a non-reference (i.e. temporary)
    .Cb
    class C {
        // ...
***************
*** 400,406
    public:
        C();
        C(int);
    !
    friend const C& operator+(const C&, const C&);
    ~C();
    C obj1;

--- 432,438 ------
    public:
        C();
        C(int);
    !
    friend C operator+(const C&, const C&);
    ~C();
    C obj1;
***************
*** 894,901
    .I delete-expression
    (_expr.delete_),
    (5) in several situations due to the handling of exceptions
    (_except.handle_).
! A program is ill-formed if the destructor for an object is implicitly used
! and it is not accessible (_class.access_).
    .ix "explicit destructor-call
Destructors can also be invoked explicitly.
    .P
--- 926,936 ------
    .I delete-expression
    (_expr.delete_),
    (5) in several situations due to the handling of exceptions
    (_except.handle_).
! ."France 10
! ."destructor must be accessible at the point of declaration
! A program is ill-formed if an object of class type or array thereof is declared and the destructor for the class is not accessible at the point of the declaration.
    .ix "explicit destructor-call
Destructors can also be invoked explicitly.
    .P
***************
*** 1338,1351
in the
.I initializer-list
than members of the aggregate,
! each member not explicitly initialized shall be copy-initialized
! (_dcl.init_) with an
! .I initializer
! of the form
! .CW T()
! (_expr.type.conv_), where
! .CW T
! represents the type of the uninitialized member.
.N[ _dcl.init.aggr_ describes how
.I assignment-expression s

--- 1373,1382 ----- in the
.I initializer-list
than members of the aggregate,
! ."Canada 5
! ."default initialization instead of initialization with T() covers array
case
! each member not explicitly initialized shall be default-initialized
! (_dcl.init_).
.N[ _dcl.init.aggr_ describes how
.I assignment-expression s

***************

*** 1550,1555
is evaluated as part of the initialization of the corresponding
base or member.
.P
If a given nonstatic data member or base class is not named by a
.I mem-initializer-id
in the

--- 1581,1588 ----- is evaluated as part of the initialization of the corresponding
base or member.
.P
+ ."Canada 9
+ ."ill-formed if no ctor-initializer and const member
If a given nonstatic data member or base class is not named by a
.I mem-initializer-id
(including the case where there is no

***************

*** 1552,1559
.P
If a given nonstatic data member or base class is not named by a
.I mem-initializer-id
! in the
! .I mem-initializer-list ,
then
.LI
If the entity is a nonstatic data member of (possibly cv-qualified)

--- 1585,1594 ----- ."ill-formed if no ctor-initializer and const member
If a given nonstatic data member or base class is not named by a
.I mem-initializer-id
!(including the case where there is no
! .I mem-initializer-list
! because the constructor has no
! .I ctor-initializer ),
then
.LI
If the entity is a nonstatic data member of (possibly cv-qualified)
***************
*** 2377,2382
if not declared by the user,
a base class copy assignment operator is always hidden
by the copy assignment operator of a derived class (_over.ass_).
.P
A copy assignment operator for class
.CW X
--- 2412,2430 -----
if not declared by the user,
a base class copy assignment operator is always hidden
by the copy assignment operator of a derived class (_over.ass_).
+ ." USA CD2-core 1-5 core-672
+ ." Using-declarations cannot introduce copy-assignment operators
+ A
+ .I using-declaration
+ (_namespace.udecl_) that brings in from a base class an assignment
operator
+ with a parameter type that could be that of a copy-assignment operator
for the
derived class is not considered an explicit declaration of a copy-
assignment
+ operator and does not suppress the implicit declaration of the derived
class
+ copy-assignment operator;
+ the operator introduced by the
+ .I using-declaration
+ is hidden by the implicitly-declared copy-assignment operator in the
derived
class.
.P
A copy assignment operator for class
.CW X
***************
*** 2444,2449
are assigned,
in the order in which they were declared in the class definition.
Each subobject is assigned in the manner appropriate to its type:
.LI
if the subobject is of class type,
the copy assignment operator for the class is used;
--- 2492,2500 -----
are assigned,
in the order in which they were declared in the class definition.
Each subobject is assigned in the manner appropriate to its type:
+ ." USA CD2-core 1-10
+ ." copy-assignment called for subobjects is never called through
+ ." virtual function mechanism
.LI
if the subobject is of class type,
the copy assignment operator for the class is used
***************
*** 2446,2452
Each subobject is assigned in the manner appropriate to its type:
.LI
if the subobject is of class type,
! the copy assignment operator for the class is used;
.LI
if the subobject is an array, each element is assigned,
in the manner appropriate to the element type;

--- 2497,2505 ------
." virtual function mechanism
.LI
if the subobject is of class type,
! the copy assignment operator for the class is used
! (as if by explicit qualification; that is,
! ignoring any possible virtual overriding functions in more derived
classes);
.LI
if the subobject is an array, each element is assigned,
in the manner appropriate to the element type;

*** ..\cpp Tue Jul 15 20:44:10 1997
***************
*** 179,185
After all replacements due to macro expansion and the
.CW defined
unary operator have been performed,
! all remaining identifiers are replaced with the pp-number
.CW 0 ,
and then each preprocessing token is converted into a token.
The resulting tokens comprise the controlling constant expression

--- 179,196 ------
After all replacements due to macro expansion and the
.CW defined
unary operator have been performed,
! ." France 15 _161/ 16.1 [cpp.cond]
! all remaining identifiers and keywords\f,
! .Fs
! An alternative token (_lex.digraph_) is not an identifier,
! even when its spelling consists entirely of letters and underscores.
! Therefore it is not subject to this replacement.
! .Fe
! except for
! .CW true
! and
! .CW false ,
! are replaced with the pp-number
.CW 0 ,
and then each preprocessing token is converted into a token.
The resulting tokens comprise the controlling constant expression

***************
*** 221,226
 .Fe
Also, whether a single-character character literal may have a negative
value is implementation-defined.
 .P
Preprocessing directives of the forms
 .Cb
--- 232,241 ------
 .Fe
Also, whether a single-character character literal may have a negative
value is implementation-defined.
+ ." France 15 _161/ 16.1 [cpp.cond]
+ Each subexpression with type
+ .CW bool
+ is subjected to integral promotion before processing continues.
 .P
Preprocessing directives of the forms
they are never scanned for macro names or parameters.

to be replaced by the replacement list of preprocessing tokens
that constitute the remainder of the directive.

The replacement list is then rescanned for more macro names as specified below.

--- 479,492 ------
they are never scanned for macro names or parameters.

to be replaced by the replacement list of preprocessing tokens
that constitute the remainder of the directive.

An alternative token (_lex.digraph_) is not an identifier,
even when its spelling consists entirely of letters and underscores.
Therefore it is not possible to define a macro
whose name is the same as that of an alternative token.

The replacement list is then rescanned for more macro names as specified below.

--- 1,4 ------
"Compatibility" diff 3 (informative)

This Annex summarizes the evolution of \*C since the first edition of

--- 1,6 ------
"Compatibility" diff 3 (informative)
+ " UK 229 11/2 C [diff]
+ .." marking the start of deleted text

This Annex summarizes the evolution of \*C since the first edition of

183,188
The bool type (_basic.fundamental_).

The subclauses of this subclause list the differences between \*C and ISO C, by the chapters of this document.

--- 185,192 ------
The bool type (_basic.fundamental_).
+ " UK 229 11/2 C.1.2 [diff.c++]
+ .." marking the end of deleted text

The subclauses of this subclause list the differences between \*C and ISO C, by the chapters of this document.
Programs and headers that reference `\_\_STDC\_\_` are quite common.

This table is reproduced unchanged from ISO/IEC PDTR 10176, produced by ISO/IEC JTC1/SC22/WG20,
This Clause lists the complete set of hexadecimal code values that are valid in universal-character-names in '*C identifiers (_lex.name_).

This table is reproduced unchanged from ISO/IEC PDTR 10176, produced by ISO/IEC JTC1/SC22/WG20,

---

** 75,78  
ff21\-ff3a, ff41\-ff5a, ff66\-ffbe, ffc2\-ffc7, ffca\-ffcf, ffd2\-ffd7, ffd8\-ffdc, 4e00\-9fa5

--- 78,84 ----

ff21\-ff3a, ff41\-ff5a, ff66\-ffbe, ffc2\-ffc7, ffca\-ffcf, ffd2\-ffd7, ffd8\-ffdc, 4e00\-9fa5

The value "5e76" above would seem to be a typo.
We should check with and/or resynchronize with WG20.