Minutes of San Jose X3J11 Meeting
6-10 June 1994

Hotel Sainte Claire
Santa Vesta Room
San Jose, California

Attendees

David Alpern, John Benito, Harry Cheng, Jerome Coonen, Frank Farance, Ron Guilmette, Doug Gwyn, Rex Jaeschke (Chair), Bob Jervis, Larry Jones, David Keaton, John Kwan, Tom MacDonald, Stuart McDonald, Randy Meyer, Dave Mooney, Esther Park, Tom Plum (Vice Chair), Linda Stanberry (Acting Secretary), Jim Thomas, Fred Tydeman, Douglas Walls, Jeff Zeeb, Jonathan Ziebell.

Legend

The following symbols in the left margin of these minutes have the indicated meaning:

A	General approval
SV	Straw vote
MSP	Moved, seconded, passed (formal vote)
*	Action item

The activities reported here are grouped by subject and do not necessarily follow the exact chronological order of presentation during the meeting.

Formal votes are reported as:

In-favor / Opposed / Abstaining / Not-voting / Total-eligible

Secretary's note: "McDonald" refers to Stuart McDonald and "MacDonald" refers to Tom MacDonald in these minutes.

1. Opening Activities

1.1 Opening comments, goals and purpose of the meeting

Jaeschke convened the meeting at 9 AM, 6 June 1994.

The goals and purpose of the meeting were announced.

- This is the first meeting since the merger of NCEG with J11, so meeting for 5 days instead of 3.
- With respect to NCEG issues, this is business as usual: wrap up the 3 groups already submitted to J11, continue with remaining 4 groups.
- New item since the Kona meeting is the new C standard charter.

1.2 Host Facilities

Benito (Perennial) and Kwan (Hewlett Packard) were the hosts, and described arrangements for duplicating and file printing. Information on local eateries was also provided.

1.3 Introduction of Participants

Attendees introduced themselves and indicated roles as focal points for NCEG subgroups or other work items. A copy of the attendance sheet is attached to these minutes.

1.4 Approval of previous minutes [N321/94-005, N315/93-062]

A The minutes from the Kona meeting were accepted as distributed.

1.5 Status of action items from previous minutes

[From the Kona J11 meeting, N321/94-005]

Gwyn agreed to provide write up on editorial changes [to Defect Report?] and post it to X3J11. Work must be done by Plauger. Done, in document.

Jaeschke [will] get resolution of LIA review. Responded on our behalf, "not rejecting, just not actively supporting."

Plum will investigate availability of machine readable X3J16/WG21 document. Sent DOS floppy to Jaeschke. Is available to standards members on floppies through Randall Swan. The discussion in J16 was that it was not to be distributed otherwise (e.g., on reflectors). Benito offered to make copies available to J11 members at this meeting.

Jaeschke will review SC21/WG3 ISO 10728 "C Binding IRDS Database." No action taken prior to deadline.

Farance will review [Portable Common Tool Environment (ECMA)] document. Done.

Farance will do electronic document pilot project. Continuing activity, will report on status later in this meeting.

Jaeschke will be a focal point for getting [revision of C standard] organized. Done, N328/94-012, proposed charter in mailing.

Plauger, Feather, and Gwyn are review committee [for edited RR and TC]. Done.

Keaton will take responsibility for [Designated Initializers and Compound Literals] document since Prosser is unavailable. Revised version in first mailing. Done.

Meissner and MacDonald will reconcile [NCEG and J11] mail reflectors. Inaction, but MacDonald reports no mail to NCEG reflector since the last meeting.

* MacDonald will post an announcement to the NCEG reflector that it will be

discontinued and future postings should be sent to the J11 reflector since the two committees have now merged. To subscribe to the J11 reflector, send to

```
x3j11-request@osf.org
```

Benito will verify that Convex is still hosting the December, 1994, meeting. Done–Convex to distribute meeting information at this meeting.

Jaeschke will talk to X3 sceretary and X3J16 about joint WG14/X3J11 meeting schedule and mechanics - how X3 feels about it. Jaeschke was informed that we cannot hold "joint" meetings, but that we can "co-locate" our meetings (sigh!). Plum noted that we should not refer to "joint committees" either as there is no such thing.

MacDonald will do WG14 mailings. Done.

Farance will investigate electonic distribution [of mailings]. Done.

Gwyn will write up exact wording specified [for defect report in N297 #2]. Done, included in the mailing.

[From the Kona J11.1 meeting, N315/93-062]

Jaeschke will email both the NCEG and J11 reflectors to notify anyone who may be affected on the rules for attendance and voting after the merger. Done.

Keaton will take over work on the compound literal/designated initializers proposal, and pursue answers to questions raised at [the Kona] meeting. Done.

MacDonald/Homer will produce revised proposal for aliasing. Done, N334/94-019.

MacDonald will recommend changing the names defined in <complex.h> to use underscores to Knaak. Done, N335/94-020.

Thomas will prepare draft #2 of the FPCE document with [Kona meeting proposed] changes. Done, N319/94-003.

Farance will investigate other languages' features (Fortran 90 and HPF) that may have an impact on the DPCE and VLA proposals. Done, will report later in this meeting on findings.

Meissner will determine what is the official dictionary for terms not defined in the C standard. No word on this as yet, and Meissner not attending. Gwyn notes that the answer to this may be relevant for one of the defect reports.

Meissner will expand and revise the fat pointer proposal for the next meeting. Farance will contact Meissner about this.

Thomas will provide his comments for his votes on the CRI VLA proposal [to MacDonald]. Not done yet.

MacDonald will forward the CRI VLA proposal to J11. Done, N317/94-001.

Kwan will revise the <inttypes.h> proposal for the next meeting. Done, N349/94-034.

1.6 Distribution of new documents

New documents were assigned WG14/X3J11 numbers, and will appear in the next X3J11 mailing. Available documents were distributed.

N348/94-033 "Extending C with Arrays of Variable Length," Cheng, distributed.
N349/94-034 "Extended Integers for C," Kwan, distributed.
N350/94-035 "An Alternative to Imaginary Types," MacDonald, distributed.
N351/94-036 "Model Electronic Document Project Status," Farance, distributed.
N352/94-037 "Minutes of San Jose X3J11 Meeting," Stanberry.
N353/94-038 "Fortran 90 VLA/Data Parallel Report," Farance, distributed.
N354/94-039 "Complex C Extensions," Thomas, distributed. (Formerly X3J11.1 93-048)
N355/94-040 "Extended Integer Range and Portable Data Structures," Farance, distributed.
N357/94-041 "Designated Initializers," Prosser and Keaton.
N356/94-042 "Compound Literals," Prosser and Keaton.
N358/94-043 "Dallas Meeting Information," Meyer, distributed.
N359/94-044 Letter to Jaeschke from William Rinehuls, distributed.
N360/94-045 "A Brief Note Regarding Defect Report Processing Procedures," Guilmette, distributed.
N361/94-046 "Call for volunteers X3J11," Arnold.
N362/94-047 "Comments on Complex vs Imaginary Types for C," Gentleman, distributed.

1.7 Approval of the agenda

Jaeschke identified papers to be addressed under each agenda topic. The agenda was slightly modified to accommodate new requests for agenda time and time conflicts for some members.

Jaeschke noted that he would like to have formal votes during the agenda time for the Designated Initializers/Compound Literals, Alisasing, and FP/IEEE subgroups to freeze these documents at this meeting. Gwyn asked for clarification on whether TR items are to become part of a revised standard. There was lots of discussion on this point: many believed the TR is a "starting point" for future standards. Thomas suggested that we need to find the right middle ground for what weight we assign to the TR. Gwyn suggested the TR could have status of POSIX-like optional extensions. Plum wanted to clarify whether the TR would have rigorous or optional status.

MSP Move that we clarify that adoption of the TR is not necessarily an endorsement of a future C standard. (Plum, Gwyn)
15/0/1/3/19

Gwyn suggested that the introductory wording of the TR should include this clarification. Thomas noted that parts of the TR are not for universal audience, and that we should recommend some parts for specific classes of implementors and users.

1.8 Identification of voting members

After the attendance list was circulated, Plum announced that there were 19 eligible voting members, of whom 16 were attending.

2. Liaison reports

WG14

Benito reported that the only new item is the need to form a position on the DAM.

WG21/X3J16

Plum noted that he will bring up additional liaison issues as they are discussed in their agenda times. There are several areas in which C++ is watching C.

Gwyn pointed out that the latest C++ working paper does not list the C standard as a base document. Plum verified this and stated that it is unfortunate from the C compatibility viewpoint.

Thomas asked if there was any interest in the C++ committee in maintaining largest compatible subset as suggested by Plauger. Plum replied that there is none at the moment; chapter 13 of the current working paper is the best attempt to provide a listing of what differs between C and C++.

Plum indicated that the Waterloo meeting in July is critical for making CD registration vote, which is the first in what is now a 3-step sequence for ballotting. Eight months after the CD-registration vote, there is a CD ballot, and eight months after that, a DIS ballot. Hence, it will be roughly 2 years plus 8 months until the final standard. This is relevant for us as we begin revision of the C standard–it is very difficult to do ISO ballotting in less than 8 months.

Farance asked if WG14/J16 is aware of our beginning work on a revised standard? Plum indicated they are not. Jaesckhe added that it is better not to inform them now, but postpone any announcement until we are fully organized and then Plauger will notify them when necessary.

X3H5

Jaesckhe reported that he had received a letter from William Rinehuls, Chairman of the OMC (used to be SPARC), notifying him that X3H5 has requested that it be changed from an active to a maintenance technical committee, and enquiring whether or not J11 is interested in assuming the responsibility of producing the C binding for the X3H5 model. OMC cannot grant the request to change the status of X3H5 until they complete publishing of their standard, including the binding for C. We need to respond to them by 10/21.

Farance suggested this be discussed under VLA or DPCE agenda time.

Gwyn asked for clarification: are we being asked to create a binding for a non-existent standard? Jaeschke indicated that there would be a standard.

MacDonald noted that the X3H5 model includes shared memory as a part of their charter.

Further discussion was deferred to other business.

X3H2

Tydeman reported that the US voted to accept LIA 1/94.

MacDonald asked if we know what the implications are for language standards. Plum believes that each committee will be tasked at the next review to critically review and justify any non-compliance. MacDonald asked do we specify what part of the C standard to use for LIA? Plum emphasized that LIA is a standard for standards committees, not for implementors. Tydeman, however, believes that an implementation must provide a LIA-compliant switch. Plum reiterated that the main impact will be on the next revision, and that a standards committee is entitled to say, "This requirement does not exist well with this language architecture."

POSIX

* Farance will review API - C Binding for File Transfer and respond on our behalf.

X3T2

Jaeschke reported that he had received notification they have a liaison with our committee.

* Jaeschke will take on task of being our liaison to X3T2.

Miscellaneous

Farance reported that he had attempted to get information from ANSI on where our money is spent with respect to the International Program Fees. What he learned is: they collect $100K-1M; very little is spent on travel; but no response was received on repeated questions on where it IS spent. Farance suggested that X3 members should pay fee since they have the most to gain and membership not available to most J11 members. Plum questioned these assumptions about the restrictions on membership.

Farance reported on the Electronic Document Project [N341/94-036]. SGML is a standard for documents. Farance will help authors put documents in SGML format. Other organizations are far behind us in investigating this area.

SV In favor of giving Farance permission to respond on behalf of X3J11 to RFI's from CBEMA on how to publish and produce electronic documents.
Lots Yes. 0 opposed.

* Farance will circulate his response to CBEMA RFI's on electronic documents to J11.

Farance has begun conversion of the RR and TC documents to SGML and will convert other documents if provided. Gwyn asked if there will be an SGML to postscript translator. Yes, Farance will provide this support for multiple UNIX

targets. Thomas asked about how to edit converted documents. Farance responded that he would not be taking on the task of doing mass editing.

3. Designated Initializers and Compound Literals [N302/93-049]

Keaton presented proposed changes to these proposals since the Kona meeting and subsequent discussions with Prosser. These changes only affect the compound literals part of the proposal.

Prosser indicated that there was no intent to imply dynamic allocation nor scope other than ordinary block scope as had been questioned at the Kona meeting. Keaton proposed changes which would clarify this.

SV Change page 2, line 8 of compound literal proposal to specify "enclosing block" rather than "enclosing function body." Delete page 3, lines 7-18.
Lots Yes. 0 opposed.

Mooney proposed that we change the example on page 3, line 21 because of conflict with proposed response to one of the defect reports regarding sharing of string literals. Gwyn and Plum suggested that we also delete next example for similar reasons.

SV Change page 3, line 21 to read:

```
(const char[]){"abc"} == "abc"
```

and delete two sentences and following example on lines 22-25.
Lots Yes. 0 opposed.

The reference on page 3, line 32 for subclause 7.5.7 should be corrected to 6.5.7.

Guilmette asked for clarification of the lvalue-ness of compound literals: would it be simpler if they were not treated as lvalues? how does this relate to Plum's concerns about C++ compatibility? Keaton clarified that compound literals behave as auto variables, not as casts.

Plum repeated concerns raised at the Kona meeting about compound literals vs. C++ constructors for creating temps. Note, there is no problem with designated initializers, nor with compound literals used in initializers at file scope. The problem is with auto temps. For constructors, need to specify interface for creating temps of structured type. The compound literals proposal attempts to solve the same problem with an incompatible syntax, as in:

```
.::member_type{ ... }
```

Gywn and MacDonald pointed out that you can't do this syntactically in C. MacDonald further stated that there is a philosophical difference here that the C extension is to be handled in the compiler without requiring the user to specify a constructor. Gwyn added that this is simpler, involving no function overhead.

It was also noted that compound literals apply to arrays, and constructors don't. The constructor syntax would also require allowing function members which

would probably not be accepted as a C extension. Plum promised to raise this objection again if J11 considers adding member functions.

Keaton asked to accept the proposed changes and freeze the amended proposal. Plum suggested that the proposal be separated into two proposals, which would require reordering/revising some examples, and/or cross-referencing the proposals. If they were separated, we can revisit objections to the compound literals proposal without affecting the designated literals proposal.

MSP Move that we separate the compound literals and designated initializers proposal into two proposals. (Plum, Gwyn)
8/5/1/5/19

Thomas asked if we could vote on freezing the proposals now, or if we would have to review the revised proposals. It was decided to appoint a review committee and conditionally freeze the proposals subject to this committee's review.

* Keaton will split the compound literals and designated initializers proposals into two proposals.

* MacDonald, Mooney, and Gwyn will review the separated compound literals and designated initializers proposals.

MSP Move that, after we make the indicated changes to the compound literals and designated initializers document, we adopt the amended documents as the final versions, subject to review by the review committee. (Keaton, Jones)
16/0/0/3/19

4. Variable Length Arrays [N317/94-001, N348/94-033]

MacDonald stated that the CRI proposal is done–taken as far as it can go. Farance indicated he had a new document that will be in the pre-Dallas mailing. Cheng has a revised document to present.

Plum reported C++ liaison issues. The C++ area that is most affected by these proposals is parts of the library, such as support for dynamic arrays. All such C++ proposals have currently been put on hold, however. The C++ array objects proposed will probably be some struct with bookkeeping information (i.e., a dope vector). As an aside, Plum encouraged DPCE subscripting to consider using a parenthesized list of comma-separated subscripts as these could be treated as overloaded operators.

MacDonald noted that F77 can be done without dope vectors but that F90 used dope vectors for slicing.

Cheng gave a whirlwind tour of his revised proposal for VLA's, N348.

- Overview
  - Deferred-Shape arrays
  - Assumed-Shape arrays
  - Pointers to array of assumed-shape
  - Rationale and Differences between this proposal and others

- Definitions of array, rank, extent, shape, and size were reviewed
- VLA type includes
  - Deferred-shape arrays
  - Assumed-shape arrays
  - pointers to array of assumed-shape (fat pointers)
  - Example showing syntax of these types:

```
int A[10][100], B[4][8], C[10], g[10];
void funct(int a[:][:], (*b)[:], c[], n, m) {
/* a: assumed-shape array */
/* b: pointer to array of assumed-shape */
/* c: incomplete array completed by function call */
  int d[4][5]; /* d: fixed-length array */
  int e[n][m]; /* e: deferred-shape array */
  int (*f)[:]; /* f: pointer to array of assumed-
                  shape */
  extern int g[];  /* incomplete array completed by
                      external linkage */
  int h[] = {1,2}; /* incomplete array completed by
                      initialization */
}
funct(A, B, C, 10, 100);
funct(B, A, C, 4, 8);
```

Note: all VLA array extensions have been implemented and tested!

- Deferred-shape arrays
  - Constraints and semantics:
    - The size of a deferred-shape array type is obtained at program execution time and the value of the size shall be greater than zero. The size of a deferred-shape array type shall not change until the execution of the block containing the declaration has ended.
    - Deferred-shape arrays shall be declared in block scope such as variables inside functions and nested functions. Arrays declared with the static storage class specifier in block scope shall not be declared as deferred-shape arrays. The behavior for declarations of deferred-shape arrays with file or program scope is undefined.
    - Pointers to deferred-shape arrays shall not be declared.
    - Deferred-shape arrays shall not be declared at the function prototype scope.
    - Deferred-shape shall not mix with incomplete array type.
    - The initializers of objects that have static storage duration are evaluated and the results are stored to objects at compilation time. But, the initializers of objects with automatic storage duration and size expression of deferred-shape arrays are evaluated and values are stored in the object at program execution time.
    - The deferred-shape array shall not be initialized.
    - For two array types to be compatible, both shall have compatible element types and the same shape.
  - Switch statement - same as CRI proposal, essentially
  - Goto statement - differs from CRI proposal because of allowing nested functions.

MacDonald clarified that he was talking about **longjmp** problems in Kona, not addressing nested functions.

- Members of structs and unions
  - **offsetof** is built in operator ; must be computed at run time if member is a deferred-shape array;.
- **sizeof** - must be computed at run time for deferred-shape arrays and structs or unions that contain deferred-shape members.
- Other data types and pointer arithmetic - same as fixed length arrays

- Assumed-shape arrays
  - Constraints and semantics
    - Assumed-shape arrays shall be declared at the function prototype scope or in a typedef declaration. The assumed-shape array is a formal argument which takes the shape of the actual argument passsed to it. That is, the arrays for actual and formal arguments have the same rank and same extent in each dimension. The shape of assumed-shape arrays cannot be determined until execution time. The rank of an assumed-shape array is equal to the number of colons in the assumed-shape specification.
    - Assumed-shape arrays may also appear in a typedef declaration.
    - Only variables of fixed-length, deferred-shape, or assumed-shape array type can be used as an actual argument of a formal argument of assumed-shape array type in function parameters. A pointer or pointer to array, which does not have the complete shape information, shall not be used as an actual argument of a formal argument of assumed-shape array type.
    - Although complete arrays can be extracted from a pointer to array, they shall not be used as actual arguments of an assumed-shape array.
    - If the operand of a polymorphic operation or function is an element of an assumed-shape array, the data type of the result and operation depend on the data type of the formal argument. However, if the formal and actual data types of an argument are different, but they are compatible, the operand will be cast to an operand with data type of the formal argument before operation takes place. If an element is used as an lvalue, the rvalue is cast to the data type of the actual arguement if they are different. In other words, elements of the actual array are coerced to the data type of the assumed-shape array at program execution time when they are fetched whereas they are coerced to data type of the actual argument when they are stored.
  - Sizeof - must be evaluated at run time for assumed-shape arrays.

- Pointers to array of assumed-shape
  - Constraints and semantics
    - When a null pointer is converted to a pointer to array of assumed-shape, a null pointer is installed at the base pointer of the assumed-shape array and the bounds of the assumed-shape array are undefined.
    - An array, including fixed-length array, deferred-shape array, and assumed-shape array may also be converted to a pointer to assumed-shape array. The base pointer to array and all bounds are stored types that do not have the array shape information shall not be converted to a pointer to array of assumed-shape.
    - For two pointer types to be compatible, both shall be identically qualified and both shall be pointers to compatible types. For two pointers to fixed-length arrays to be compatible, both shapes of array pointed to by the

pointer shall be the same and the shapes shall evaluate to the same value at program execution time.
- Function prototype scope
  - A pointer to assumed-shape array can be used as an argument parameter of a function to pass arrays of different size to the function.
- Typedef - can use assumed-shape arrays in typedefs.
- Same as pointers to fixed-length arrays

- Rationale and Differences between this proposal and others
  - No deferred-shape arrays allowed at block scope with static storage duration
  - Differences between deferred-shape arrays and CRI's VLA's
    - OK in CRI proposal, not in Cheng proposal
      - deferred-shape arrays at function prototype scope
      - pointers to deferred-shape array
      - deferred-shape array mix with incomplete array type
    - Not in CRI proposal, OK in Cheng proposal
      - deferred-shape arrays as members of structures and unions, new semantics for sizeof, typedef, offsetof.
      - deferred-shape arrays in nested functions, structures and unions with members of deferred-shape arrays, new semantics for goto, obsolete features for setjmp(buf) and longjmp(buf).
      - deferred-shape arrays with static storage duration at file or program scope as vendor extension.
    - Order of evaluation defined in CRI proposal, undefined in Cheng proposal for side effects in deferred-shape array declarations:

```
int n;
int a[n++], b[n++];
```

  - Pointers to assumed-shape arrays (PASA) versus fat pointers (FP)
    - PASA - implemented and tested. FP - conceptual proposal.
    - PASA - complete similarity between pointer to fixed-length array, integrated with deferred-shape arrays and assumed-shape arrays. FP - details are not specified.
    - PASA - in nested functions. FP - unknown.
    - Syntax differences - ? vs :
    - Sizeof: PASA - same as pointer to fixed-length array. FP - the total number of bytes of the array pointed to.
    - If a null pointer is assigned to the pointer, the bounds of the array are: PASA - implementation-dependent. FP - set to zero.
  - Assumed-shape arrays (ASA) versus pointers to assumed-shape array (PASA)
    - Sizeof: ASA - the total number of bytes of the array pointed to. PASA - the same as pointer to fixed-length array.
    - Anti-aliasing: PASA - can be used with **restrict**. ASA - can't.
    - Passing array of different data type to functions. Any other typed languages - can't. PASA - can't. ASA - can.

- Implemented as C/Unix shell interpreter.

Farance presented his APL/VLA proposal.

- Features
  - shape
    - varying size arrays
    - shapeof(x) construct

- shapeis(x)
- shapeis(?)
- layout
  - layoutof(x)
  - layoutis(x)
  - layoutis(?)
  - near, far, huge ptrs
- selectors
  - slicing
  - scatter-gather
- parallelism
  - scalar/array promotion
  - iterators

- Methods of passing information (arguments) to functions

Layout	Shape	Ptr/Value	
–	–	P	Pointer
–	–	V	Value
–	S	P	VLA
–	S	V	ALO
L	–	P	Pointer to distributed value
L	–	V	Distributed value
L	S	P	Distributed VLA
L	S	V	Distributed ALO

```
int f(int n, int a[n])
             int a[?]
             int a[:]
             int a[*]
```

- shapeof(x)
  - returns array of size_t

```
int A[3][4];
shapeof(A) == { 3,4 }
int C;
shapeof(C) == null array => scalar
```

- rankof(x) => # of dimensions

```
int A[3][4];
int B[4];
int C;

rankof(A) == 2
rankof(B) == 1
rankof(C) == 0
```

- C arrays vs. ALO's (Array-Like Objects)
  - No differences except ALO's stand alone or become first class objects
  - So can freely cast one to the other
  - Give programmers what they know
  - New type qualifier "alo"

```
int B[10];
```

```
int alo C[10];

B + 1   <- same as &B[1]
C + 1   <- adds one to each element
(alo)B + 1   <- adds one to each element
(carray)C + 1      <- &C[1]
```

- Argument passing
  - passes by value – i.e., a copy of whole array
  - note in example, shape is not part of arg since shape is fixed

```
int f(int alo a[3])
{
   int sum, i;
   sum = 0;
   for (i=0; i<3; i++)
      sum += a[i];
   return sum;
}

main()
{
   int alo b[3];
   ...
   f(b); /* Passes whole array */
}
```

- Reshape - used in casts
  - allows dynamically determined shape

```
shapeis(x)
int A[12]
(shapeis({3,4}))A
```

  - reshapes array to 3x4
  - if too few elements, wrap around to beginning
  - if too many, take first N elements

Guilmette asked if this compares to `(int[3][4])A` which statically determines fixed rank shape and type. Yes.

Keaton asked what happens to contents during reshape. Constructs new ALO value.

- Shape in declaration

```
int A[3][4];
int (shapeis(shapeof(A))B;
```

  - creates B with same shape as A

- Qualifiers to **shapeis**–behavior under **const** and **volatile** qualifiers

```
int shapeis(x) Y;
   /* can change shape */
int (const shapeis(x)) Y;
   /* shape doesn't change */
int (volatile const shapeis(x)) Y;
   /* shape doesn't change; shape checking on
      assignment */
int (volatile shapeis(x)) Y;
   /* array bounds checking */
int (const volatile shapeis(x)) Y;
   /* array bounds checking; shape doesn't change */
int (volatile const volatile shapeis(x)) Y;
   /* array bounds checking; shape doesn't change;
      shape checking on assignment */
```

MacDonald suggested {:^) that if **restrict** is added, it would mean that no else has this shape!

• Interaction of **shapeis** and array declaration

```
int A[3][4];

   /* equivalent to */

int (const shapeis({3,4})) A;
```

• Unknown shape

```
f(int a[?][?])
   /* Passes pointer + shape (fixed rank) */

f(int shapeis(?) a)
   /* Passes pointer + shape (varying rank) */
```

• Layout – data distribution
  • block + scale layout descriptors defined elsewhere

```
layoutof(x) - size of extents
shapeof(layoutof(x)) = shapeof(x)

int alo by(7) x[10];
layoutof(x) = { 4,4,4,4,4,
                4,4,4,4,4 }

&x = { 1000, 1004, 1008, 1012, 1016,
       1020, 1024, 2000, 2004, 2008 }
layoutof((void)x) = { 28, 12 }
&(void)x = { 1000, 2000 }
```

Gwyn pointed out that you can't take the & of a cast to void. Farance said this would be an extension.

• layoutis operator

```
int alo /* layout spec */ x[100];
```

```
int alo layoutis(layoutof(x)) Y[100];
```

• similar to redistribute in HPF

MacDonald asked what is being redistributed? Copies object, doesn't distribute in place as HPF does.

```
f(int layoutis(?) a[100])
        /          \
 passes layout      passes ptr

/* Distributed VLA */
```

• Near, Far, Huge ptrs
  • Near:

```
int *P;   /* local ptr */
```

  • Far:

```
int layoutis(...) *P;
```

    • fast increment (local part only)
    • can NOT walk across extents
  • Huge:

```
int (volatile layoutis(...)) *P;
```

    • global ptr - slower
    • CAN walk across extents (incr works)

• Summary of shape + layout features

```
shapeof(x)
shapeis(x)
{ const/volatile } shapeis(x)
shapeis(?)
[?]
alo
carray
rankof(x)
layoutof(x)
layoutis(...)
layoutis(?)
```

• Missing features/issues
  • explicit ptr + shape paste in proto

```
f(int n, int a[n])
    /       \    \
  shape     ptr   \
                  paste
```

Resolution: will add in next revision
  • unordered args:

65

```
f(int a[n], int n)
```

Resolution: will not add this or Stallman proposal

Jaeschke asked if the status quo is the CRI VLA paper. MacDonald answered Yes, since that is the only one that is finished. Jaeschke asked if we want to continue to have VLA as part of the TR. Need to get sense of committee on continuing work on each proposal.

Cheng asked for feedback on how he could further his proposal. MacDonald suggested removing nested functions and polymorphic stuff. Gwyn added that he should concentrate on standard C and arrays. Thomas suggested that Cheng might want to present as extensions to the CRI proposal. Cheng pointed out that he had merged features (FP) from Meissner's proposal–i.e., ASA's and FP's are not in the CRI proposal.

Plum expressed that it is hard to vote. We have heard N proposals. Need to know strategy of the committee before voting. Gywn asked if we would publish a merged proposal or separate ones. Plum asked if we would include rationale on others.

MacDonald reiterated that he is done. He has no more time to spend on this proposal.

Farance is willing to host VLA meeting to facilitate merging proposals.

Cheng observed that his and the CRI proposal are much closer, the Farance proposal is much different than the others.

Plum noted that the CRI proposal has been around for some time, it's implemented and in use. MacDonald added that Stallman also uses it.

Thomas believes that in the time frame allowed, it is unlikely to get other than the CRI proposal in TR.

Jaescke called for straw poll of voting members:

SV In favor of the CRI proposal being the official VLA proposal in the TR.
9 Yes. 2 No. 5 Undecided.

Farance stated that he wants to continue his VLA proposal as an alternative to the DPCE proposal.

Plum wants to be sure that the adopted proposal could be implemented as a class with a dope vector. Is it mandatory that it be implemented as a single pointer? MacDonald replied that the CRI VLA is compatible with void * so therefore only a pointer. Farance indicated that, except for iterators, APL/VLA has all been implemented in C++ by Farance, Inc.

We then voted as if it were a formal vote to determine if any work still needed to be done.

SV In favor of the CRI proposal being the official VLA proposal in the TR.

66

11 Yes. 3 No. 2 Would not vote.

Jaeschke then called for a formal vote. Thomas asked if this can be undone or edited. Yes, we can always change our mind.

Farance expressed concern that he had moved to VLA because he was unhappy with DPCE. Gwyn suggested that J11 would consider continued work as useful to parallel proposals.

MSP Move we forward the CRI VLA proposal as part of the TR. (MacDonald, Stanberry)
11/3/0/5/19

Farance asked that the following comment be recorded in the minutes:

> It is unfortunate that there now lacks a formal forum for APL/VLA work.

Farance announced that he will sponsor a meeting 8/22-23 to work on extensions to VLA proposal. Gwyn, MacDonald, and Farance expressed interest in attending. Jaeschke suggested that Farance should announce the meeting on the J11 email reflector.

Cheng indicated that he has already tried to integrate his proposal with standard C and other VLA proposals, so unless there are specific suggestions from the committee for changes, he will just continue on his own.

Farance also asked for feedback from the committee. Would it help to break his APL/VLA proposal into 3 or 4 separate proposals, to allow the committee to pick and choose? MacDonald was in favor of this since some will have more cost than others. Would C++ compatibility description help? Gwyn noted that you can't do syntax extensions through class library.

There was discussion of whether other proposals could also become part of the TR. Jaeschke suggested we should get the sense of the committee on this–e.g., do we give agenda time for these proposals? MacDonald said we must be careful not to mislead anyone.

Jaeschke asked if there was any objection to giving agenda time? Gwyn noted that this should be conditional on having papers in the mailings. Thomas also noted that preference should be for other issues, and these should be discussed only as time allows.

5. Complex [N335/94-020, N339/94-024, N350/94-035, N354/94-039]

MacDonald reported that the status of this subgroup is still trying to decide between two proposals, differing mostly in whether or not to include an imaginary type. CRI has a revised proposal that resolves differences between their earlier proposal and one from Vermeulen for C++, and a paper on alternatives to imaginary types.

Thomas has new rationale papers for including imaginary types, and suggests that we decide at this meeting on one of the complex proposals to be forwarded.

MacDonald established the goals of this agenda time to determine direction for the subcommittee on the issue of adding an imaginary type. Knaak will edit the proposal if no imaginary type, and Thomas will edit the proposal if there is an imaginary type.

MacDonald presented changes in the CRI complex proposal, N335.

- Goal
  - Resolve differences with Alan Vermeulen's C++ proposal
  - Provide compatibility with complex classes in C++
  - Enable programs to work with both languages

- _STD_COMPLEX
  - A macro defined in <complex.h> header
  - Allows programs to distinguish these new types
  - Identical between C and C++

- New Type Names
  - float_complex
  - double_complex
  - long_double_complex

- Portability
  - Use type names instead of keyword types
  - Use the CMPLX macros instead of the "i" suffix
  - Use CMPLX macros and "creal" and "cimag" functions to convert a complex value from a higher precision to a lower precision type
  - For complex math functions, add macros such as:

```
#if defined(_STD_COMPLEX) && defined(__cplusplus)
#define csin sin
#endif
```

Plum noted that in C you can get conversion automatically across assignment but it is ambiguous in C++.

There was some discussion of using overloaded functions instead of macros to redefine the math functions. Consensus was that this would be nice if the next revision of the C standard includes overloading. Thomas and Plum suggested that if overloading is implemented, use the versions in <fp.h> or add words that the macros that map complex functions to the right names are only available if overloading is not available.

Thomas presented an overview of his proposal for complex, N354.

- Complex C Extensions
  - Spirit of FPCE
  - Competing proposal–Knaak: Complex Extensions to C (N335)
  - Modern vs. classical
  - Several secondary differences, but ...
  - One key difference: imaginary types
  - Very substantive issue

- Modern vs. Classical

- Modern
  - Models mathematical concepts
    $i^2 = -1$
    real * i -> imaginary
    real + imaginary -> complex
    symmetry between real and imaginary
    real * complex, imaginary * complex are scalar multiplications
  - More natural from mathematical & OOD points of view

- Classical
  - Motivated by data representation
    complex <--> array of 2 reals
  - Limited by data representation
    imaginary values are second class
    real * complex, imaginary * complex require complex multiplications
  - More in Fortran tradition

Gwyn questioned why the * is called scalar in modern and complex in classical. MacDonald stated that the issue is introduction of 0's that happens in classical *'s.

- Semantic/Efficiency Problem of Classical Approach
  - Modern
    `2.0i * (∞ + 3.0i) => -6.0 + ∞i`
    `2 *'s`
  - Classical

```
2.0i * (∞ + 3.0i)    => (0.0 + 2.0i) * (∞ + 3.0i)
                     => (0.0*∞ - 2.0*3.0)
                                + (0.0*3.0 + 2.0*∞)i
                     => NaN + ∞i
4 *'s, 2 +'s
```

(even real * i would require 4 *'s, 2 +'s)

MacDonald reminded the committee that it is only a problem for NaN's, infinities, and possibly signed 0's, and argued that in the complex plane, NaNs and infinities essentially give same the same information, that is, that something exceptional happened. Gwyn and Coonen disagreed: infinity is better than NaN since it is a real number.

- IEEE Compatibility
  - Modern
    Natural compatibility with IEEE arithmetic
    Completeness
    Consistent treatment of special values (infinities, NaNs, and signed zero)
    Major features of IEEE
    IEEE is dominant standard
  - Classical
    Fundamentally incompatible with IEEE arithmetic
    Acceptable efficiency prohibits completeness
    Special values must be excluded from model
    Reasonable behavior depends on optimization

Gwyn suggested that this could be handled by special rules for introduced values, reducing the answer when appropriate: behave as if an imaginary type, but don't have one. MacDonald said this is really difficult, even though you want your optimizer to give this behavior. Plum observed that IEEE doesn't have a representation to map an imaginary exactly onto the complex plane because there is no way to represent "exactly 0" which is what optimizers have to do. Coonen appealed for us not to drift into arguing virtues/faults of IEEE specifications.

McDonald pointed out that Thomas' proposal allows keeping imaginary results imaginary rather than having to coerce them to real. MacDonald argued that exceptional cases are so rare that it's not too much to ask the user to deal with them. Thomas and Coonen replied that means you have to deal with exceptional cases all the time, even though they occur only rarely, which is an undue overhead. MacDonald countered that there are only a few places where checking needs to occur. Gwyn added that IEEE rules allow delaying of some checks.

- Efficiency
  - Modern
    - Natural speed efficiency
    - Supports overloading on imaginary types
    - Natural space efficiency
  - Classical
    - Speed efficiency depends on optimization
    - Space efficiency requires implementation from reals

On the issue of overloading for imaginary types, MacDonald suggested why not just use real types and then multiply by i?

- Simplicity
  - Modern
    - More types require more language specification
    - Basic implementation is more complicated
    - Simpler for users
      - Better model for mathematics
      - Compatibility between real and complex domains
      - Predicatable behavior of special cases
      - Flexible
      - Natural efficiency
    - Supports classical programming model
  - Classical
    - Simpler language specification
    - Requires more optimization
    - More complicated for users

Final arguments were made for choosing between complex with or without an imaginary type.

Gwyn stated that, from a scientific point of view, an imaginary type simplifies tasks for users. Can be done with reals, but user must track which reals are actually imaginary. So from users' view, would prefer to have imaginary types.

Plum gave the C++ liaison point of view. There has been increasing compatibility between the C++ and C proposals, so thanks. It is doubtful that the C++ proposal will get into the standard. Plum is convinced that an imaginary type class

implementation could overlay a complex class implementation. There is still a problem of what to do with I. But he believes it is better to provide needed user functionality over inconveniencing implementors who don't believe in it.

Keaton asked if without an imaginary type is there anything that tells an optimizer when it's ok to change IEEE 0 into a true 0? For example, when IEEE 0 is an operand of *, ok to produce true 0? There was considerable discussion on this: are imaginary types only necessary for IEEE implementations? There was not general agreement, citing the document from Gentleman, N332, and others.

MacDonald argued that the cost of adding 3 x 2 = 6 new types is too much.

McDonald stated that there is an overwhelming need for some complex type so doing nothing would be the worst decision.

Keaton suggested taking a vote on requiring imaginary type only for IEEE implementations. Kwan believes need for imaginary type will go beyond IEEE so doesn't want to separate them.

SV In favor of having an imaginary type. (Voting members only)
6 Yes. 2 No. 7 Undecided.

SV If a formal vote, would you be in favor?
10 Yes. 4 No. 1 Would not vote.

MSP Move that we proceed with proposal for complex with imaginary type. (Thomas, Gwyn)
10/4/0/1/19

MacDonald asked that the following be recorded in the minutes with CRI's No vote:

> "The additional complexity of adding imaginary types isn't warranted by the benefit users will see."

Plum asked if any member wished to record comment with No vote. No one else did.

* MacDonald, Tydeman, Plum, and Keaton will serve as review committee for the proposal.

Jaeschke noted that, since there is little time left to complete this proposal by the end of this year, that it is possible to get a futher extension for the TR to allow sufficient public review of this and other proposals that are finishing up. We will need to make such a decision in December.

6. Aliasing

MacDonald reported on the status of the CRI restricted pointer proposal. At the Kona meeting, there was discussion of a proposed edit which was revisited.

Restricted pointer edit

- Revise old section 1.6: Aliasing of unmodified objects
- Move it and two other sections on design choices into an appendix.

• In the Semantics subsection of section 2, delete the "If O is modified, then" phrase, leaving "All references to values of O shall be through pointer expressions based on P."

Review: the intent

Consider types with two successive restricted pointer derivations, as in:

```
int * restrict * restrict p;
int * restrict * restrict q;
```

The intent was that p and q can be analyzed as if they were arrays:

```
int p[][n];
int q[][m];
```

Review: the problem

```
void f1(int n, int * restrict * restrict p,
               int * restrict * restrict q)
{
   int i,j;                        /* without edit, */
   for (i=0; i<n; i++)             /* optimization  */
      for (j=1; j<n; j++)          /* is inhibited. */
        p[i][j] += q[i][j-1];
```

Without the edit, p and q may point to the same array of restricted pointers, since that array is not modified.

• The most natural usage does not promote optimization.

Review: VLA example

```
void f2(int n, int (* restrict) p[n],
               int (* restrict) q[n])
{
   int i,j;                        /* Optimization */
   for (i=0; i<n; i++)             /* is easy.     */
      for (j=1; j<n; j++)
        p[i][j] += q[i][j-1];
```

Here, since p and q point directly to the modified objects, those objects must be different (with or without the edit).

• The original intent was that `f1` should be no more difficult to optimize than `f2`, and this intent is realized by the edit.

Review: a disadvantage

The disadvantage is that function calls that result in aliasing of unmodified objects will now have undefined behavior, even in those cases in which no optimizations are at stake.

```
void f(int n, int * restrict p, int * restrict q,
```

```
                                        int * restrict r)
{
   int i;
   for (i=0; i<n; i++) p[i] = q[i] + r[i];
}
```

• A call `f(n,a,b,b)` will be undefined (even though it will almost certainly give the expected results).

A solution

`const` and **`restrict`** can be used together to solve this problem by making the appropriate aliasing assertion:

```
void f(int n, int * restrict p, int * const q,
                                int * const r)
{
   int i;
   for (i=0; i<n; i++) p[i] = q[i] + r[i];
}
```

• The prototype alone asserts that `p[*]` is not aliased by `q[*]` or `r[*]`.
• This allows the loop to be optimized with minimal analysis.
• `f(n,a,b,b)` has defined behavior.
• Could introduce block scope pointers for incrementing (instead of indexing).

Issue: read-only objects not affected by **`restrict`**.
Goal: remove wording making behavior undefined if modified. Intent was always to treat as if arrays.
Dilemma: ok if restricted pointer to array, but not ok if restricted pointer to restricted pointer.
Resolution: use **`const`** pointers instead of restricted pointers for read-only arguments to functions.

Plum raised two issues: (1) What happens in the function? and (2) What does the prototype tell you? (In C++ prototype, the **`const`** means nothing–much argued over, but finally agreed that it is of no interest to the interface.) MacDonald clarified this is (1) definition vs. (2) prototype distinction: **`const`** tells the compiler what to do with definition with respect to optimization. Plum suggested that one could also use **`const int * const`** to further indicate the read-only nature of the argument. Gwyn asked whether one could still give up exclusive access to a restricted pointer argument. MacDonald clarified that this could be done within the function.

MSP Move we adopt N334/94-019, which includes this edit to restricted pointers, as the final aliasing component of the NCEG TR. (Plum, MacDonald)
16/0/0/3/19

Plum made an appeal for help for strategy to get C++ to adopt this proposal. MacDonald can get volunteers come give proposal presentation to the C++ committee.

7. FP/IEEE [N319/94-003, N320/94-004, N338/94-023]

Thomas led the discussion on the FPCE proposal. His goal was to (1) get the proposal approved as a TR, (2) clarify significance of TR, (3) see if there are any remaining FPCE issues, and (4) get advice on where to go from here with FPCE.

There have been no changes to the proposal document since the last meeting, so what issues remain?

MacDonald has already documented his issues with the proposal document.

Plum asked for clarification on several points. Since this is clearly aimed at an IEEE audience, are there parts of the proposal where IEEE is not required? Yes, it is a 2-level implementation specification. What parts require IEEE? Nothing in the basic proposal, and it is clearly documented where IEEE is required.

What about those extra relational operators? These had been presented to the C++ committee, and not warmly received. They are there to support partial ordering.

Thomas indicated this was addressed in the rationale presented in N338: achieve predictable behavior with exceptional values; user does not have to be conscious of exceptional values flowing through the code. He gave examples of a typical problem for which the extra operators were useful, and indicated what the alternatives would be without the extra operators:

Using C operators:

- `if (fabs(x) < t) return x; else ... ;`
  but want to return x if x is a NaN, so
- `if (fabs(x) >= t) ...; else return x;`
  but this raises an exception to protect against code that is not NaN-aware
- `if (isnan(x) || fabx(x) < t) return x; else ...;`
  awkward, inefficient - costs an extra test

Macro solutions:

- ```
  if (fpcompare(fabs(x),FP_LESS|FP_UNORD,y))
      return x; else ...;
  ```
- `if (iscmp_lt_un(fabs(x),y)) return x; else ...;`
  awkward, efficiency questionable

New operators:

- `if (fabs(x) <? t) return x; else ...;`
  conflicts with trigraphs?
- `if (fabs(x) !>= t) return x; else ...;`
  efficient!

Plum asked how is efficiency guaranteed? what is meant by efficiency?

Coonen stated that most of the extra operators are implemented by a single compare and branch operation. Another measure of efficiency is that these operators make programmers more efficient and make more maintainable, more robust code.
  ```

Gwyn was skeptical of how these operators would free users from NaN-awareness. He argued that you need to think about NaN's at the beginning of each library function, check parameters before comparisons, and then know the algorithm will work for the body of the function, so isnan() is sufficient.

Plum made another objection to the new operators: there is really no way to distinguish whether written code has been "robustified" since sometimes the C operators are correct. isnan() doesn't help. An alternative would be to require optimization of "(isnan(a) | a relop ...)" to underlying compare and branch operation.

MacDonald complained that the new operators don't help programmer productivity, that they are allowed where it is foolish (although not harmful), and that the precedence level of the new operators is different from the non-!'d operators.

SV In favor of endorsing the new relational operators from the FP/IEEE proposal.
6 Yes. 9 No. 5 Don't know/don't care.

Thomas reminded us that we have previously voted these operators out of the document, and then voted them back in.

We repeated the straw vote, asking for members to vote as if it were a formal vote (i.e., no abstentions), to determine if the inclusion of the new relational operators was a show-stopper.

SV In favor of freezing the FP/IEEE document as is (with the new relational operators).
15 Yes. 3 No. 2 Not voting.

Thomas asked that before we take a formal vote, should clarify who is served by removing the new operators. He wants some reassurance that the desired optimizations will still be possible without them. There was considerably more discussion on the possibility of replacing them with macros, focusing on function call overhead vs. optimization overhead that could be involved.

Plum argued that an implementation with macros in <fp.h> would be more likely to have wider use/acceptance. He pointed out that there are better known mechanisms for implementing in-line versions of functions for optimizations.

Plum suggested requiring macros in addition to the operators. Mooney noted that would violate the "one way" to do things.

Coonen asked what advantage macros have over operators with respect to "real estate." MacDonald stated that macros are optional to implement, but syntax is not; if NaN's not supported, the macros can be defined in terms of other operators.

Gwyn suggested that the programmer would have more flexibility to provide portability to specific environments if macros were not standardized.

Jaeschke noted that if we support two different ways to implement, already indicates a compromise.

Gwyn asked if it was just the awkwardness of the macros that was objectionable? Thomas stated that the major concern is to guarantee efficiency. Consensus was that this is as easy to guarantee with macros as with operators.

Reluctantly, Thomas agreed to present macros as an alternative to the new operators, and to outline the needed changes to the document as a result.

- There are two issues:
  - Be able to test, and
  - No exception raised if NaN generated
- Add macros like C relationals, except quiet:

```
isless
islessequal
isgreater
isgreaterequal
```

and, also quiet:

```
isunordered
islessgreater
```

- Near match with IEEE identified useful comparison operators:

IEEE Identified	FPCE Current	FPCE Proposed
`x ? y`	`x !<>= y`	`isunordered(x,y)`
`x <> y`	`x <>y`	`islessgreater(x,y)` - quiet only
`x <>= y`	`x <>= y`	`! isunordered(x,y)` - quiet only
`x ?> y`	`x !<= y`	`! islessequal(x,y)`
`x ?>= y`	`x !< y`	`! isless(x,y)`
`x ?< y`	`x !>= y`	`! isgreaterequal(x,y)`
`x ?<= y`	`x !> y`	`! isgreater(x,y)`
`x ?= y`	`x !<>y`	`! islessgreater(x,y)`

Proposed changes by section:

- §3.3.2
  - Remove entire section up to For IEEE Implementations
  - Change the For IEEE Implementations part to be

    Each of the relational operators—<, <=, >, >=—yields 0 and raises the invalid exception if its operands are unordered, that is, if one or both of the operands is a NaN.

    This is as required by the IEEE standards. The IEEE standards identify need for a total of 26 comparison predicates. The comparison macros in §4.3 supplement the Standard C equality and relational operators to address this need.
- §4.3 P33 L42. Add (for example)

  The macro

```
isunordered(arithmetic-expr, arithmetic-expr)
```

evaluates to an int expression that is nonzero if and only if its arguments are unorderd.

The macro

```
isless(arithmetic-expr, arithmetic-expr)
```

evaluates to an int expression that is nonzero if and only if its first argument is less than its second argument. The return value of `isless(x,y)` is always equal to `x<y`; however, unlike `x<y`, `isless(x,y)` does not raise the invalid exceptions when `x` and `y` are unordered.

Gwyn raised the question of how to write these macros for any type. Will this require conversion to a common type and could that generate the exception we are trying to avoid? Not if to a wider type–that conversion can not raise exception.

Guilmette asked if these macros can be used in initializers for file scope objects. No, not guaranteed to expand to constant expressions (e.g., could expand to the new relational operators, if those were implemented).

Mooney and Jaeschke suggested they could be functions instead of macros. Advantages: no multiple side effects for operands. Disadvantages: have to specify argument type!

Jaeschke and Plum believe that something must be said about possible multiple side effects. Plum suggested could create a hidden function to implement safe macro (no multiple side effects). That is, cast argument to long double and call function to implement macro. Then, according to §4.1.6, the operands are evaluated only once. Needed for other macros in <fp.h> as well?

• §4.3 P34 L19.

Add:

The translator should recognize the comparison macros (`isless`, `islessequal`, `isgreater`, `isgreaterequal`, `isunordered`, `islessgreater`) and implement them to be as efficient as if they were built in operators.

Typical IEEE hardware will support efficient implementation.

Move rationale from §3.3.2 to here.

Add rationale for removing operators.

Add table showing near match with IEEE identified relationals and say why the difference doesn't matter.

Show trivial macro definitions for implementations without NaNs, e.g.

```
#define isunordered(x,y) 0
#define isless(x,y) ((x)<(y))
```

• §A.2, A.4

Remove sections, adjust numbers, fix up references

• §B.3 Relational operators

...

`x<y -> isless(x,y)` (and similarly for <=, >, >=). Though equal these expressions are not equivalent on IEEE implementations if `x` or `y` might be a NaN, and the state of `fenv_access` is on. This transformation, which would be desirable if extra code were required to cause the invalid exception for unordered cases, could be performed provided the state of `fenv_access` is off.

...

Example

...

nor to

```
if (isgreaterequal(a,b)) g(); else f();
/* calls f without raising invalid if a and b
   are unordered */
```

nor, unless the state of `fenv_access` is *off*, to

```
if (isless(a,b)) f(); else g();
/* calls g without raising invalid if and b
   are unordered */
```

...

• §C.3

Include macros.

• §F.9.2 The fmax function

...

fmax might be implemented as

```
isnan(y) ? x : (islessequal(y,x) ? x : y)
```

...

Some applications may be better served by a *max* function that would return a NaN if one of its arguments was a NaN:

```
isnan(y) ? y : (isgreater(y,x) ? y : x)
```

Thomas asked for help with rationale for these changes. Plum suggested:

• To make it easier to implement in C++ and therefore encourage C++ implementations.

• To make better acceptability in the market place, and hence encourage more vendors to implement.

Coonen expressed that everyone loses if macros not efficiently implemented in sub-optimal compilers. It was noted that the operators will still be in the rationale, and appendix. Keaton added that the macros describe the architecture, and operators are an allowed implementation of that architecture.

Coonen noted a nice result of these changes is these macros can actually reduce multiple tests to one.

MSP Subject to editorial review, move that we freeze the FP/IEEE proposal after these proposed edits. (Thomas, Farance)
16/0/0/3/19

* Meyer, Tydeman, Kwan, and Keaton will review the edited document.

Tydeman asked whether wording under 4.2.1.2 about translation-time vs. execution-time conversions should be for only when limits are not exceeded. After discussion of possible alternatives, however, objection was withdrawn.

Gwyn noted that on pages 8 and 10 and possibly other places, "should" should be replaced with "shall" since "should" has no meaning in the C standard. Thomas noted that "should" is defined on page 4 to have the desired meaning for the proposal.

Guilmette suggested that the document be separated into (1) a binding, and (2) a set of extensions, where (1) could be further subdivided into (a) binding for IEEE and (b) binding for non-IEEE. Thomas agreed this would be possible, but Jaeschke believes that as a working document, it is best bundled as one.

Thomas noted that he had also received a request for an implementor's guide. N338 identifies what are extensions in a more succinct format.

8. Extended Integers [N349/94-034, N355/94-040]

Kwan brought a revision of the <inttypes.h> proposal to the meeting, reflecting the input received at the last meeting. Looking for more input with the hope of resolving to a final proposal by December.

Plum asked about the liaison status with C++. Kwan indicated that he had provided a copy of the proposal to Lenkov.

Kwan presented an overview of the revised <inttypes.h> proposal.

- Purpose
  - Make C the programming language of choice
  - Provide a way to write portable code by:
    - Defining new integer types
    - Providing consistent properties and behavior
    - Easily implemented
  - Takes a minimalist approach

- <inttypes.h>
  - Integers of exactly n bits

typedef ? int8_t
typedef ? int16_t
typedef ? int32_t
typedef ? int64_t
typedef ? uint8_t
typedef ? uint16_t
typedef ? uint32_t
typedef ? uint64_t

- Largest integer supported
  typedef ? intmax_t
  typedef ? uintmax_t
- Other types
  - Most efficient integer type
    typedef ? intfast_t
    typedef ? uintfast_t
  - Integer data types that are large enough to hold a pointer to void (*void)
    typedef ? intptr_t
    typedef ? uintptr_t

  Note: not all pointers are the same size in some systems

Plum remarked that it would be desirable to have a feature test to ask whether there is such a type; make it a property of these types, if feature exists, that you can assign to and then get back the same pointer. MacDonald suggested that these could then be macros instead of typedefs. Plum stated C++ would argue that they should be typedefs.

Gwyn also noted that there also were no feature tests for the "at least" types (because of upper bound of 2n?), and may not be able to implement them as specified. Would prefer "smallest type of at least n bits (period)." Keaton added that you need "smallest, most efficient" to prevent implementations using shift and mask.

Plum asked if all the underlying types must be native. Yes.

Keaton recalled that we wanted "most efficient" types in the proposal. Jones said Yes, but not necessarily "a" most efficient type. Kwan noted that intfast_t and uintfast_t are supposed to be these types. The "exactly" and "at least" types are not necessarily most efficient.

Mooney asked how does the user choose? Use intfast for algorithms and others for portability. Gwyn believes these are orthogonal issues; the Farance proposal will address "fast n" types.

- Limits
  - Fixed length integer type

```
#define INT8_MIN     (-128)
#define INT16_MIN    (-32767 - 1)
#define INT32_MIN    (-2147483647 - 1)
#define INT8_MAX     (127)
#define INT16_MAX    (32767)
#define INT32_MAX    (2147483647)
#define UINT8_MAX    (255)
```

#define UINT16_MAX (65535)
#define UINT32_MAX (4294967295)
- Implementation defined limits
#define INTMAX_MIN ?
#define INTMAX_MAX ?
#define UINTMAX_MAX ?
#define INTFAST_MIN ?
#define INTFAST_MAX ?

Kwan noted that these macros serve a dual role as they can also be used as feature tests; if the corresponding type is not implemented, the macro is not defined.

Gwyn believes these are incomplete, other feature tests are needed.

Jaeschke asks what is the type of the macros–needs to be specified. Also need conversion rules.

Plum asked what if that type can't be stated in portable C.

Must be consistent with responses to DR earlier this week (#69?).

- Integers of "at least" n bits
  - Signed integral types of at least n but less than 2n bits:
    typedef ? int_least8_t
    typedef ? int_least16_t
    typedef ? int_least32_t
    typedef ? int_least64_t
  - Unsigned integral types of at least n but less than 2n bits:
    typedef ? uint_least8_t
    typedef ? uint_least16_t
    typedef ? uint_least32_t
    typedef ? uint_least64_t

- Formatted I/O
  - Extend the "width" specifier
    "wnd" where n is the width of object in bits
    printf("int 16 is %w16d\n",s16);
  - Use * as the "width" and sizeof operator
    printf("int fast 16 is %w*d\n", sizeof(intfast16_t)*bits_per_byte,
    myint);
    can be used with data types of "at least" n bits
  - Requires no extensions
    use predefined macros
    always cast to intmax_t

Gwyn and MacDonald believe all compilers can support the PRI macros, but not the SCN macros, which leads to a portability problem. Plum suggested that could scan into a native type and then use feature tests to determine which type to use to assign result. Jones and Gwyn stated this speaks for using intmax_t. Plum thinks this would be inefficient for 8, 16, and 32 macros.

Gwyn thinks this is too much complexity when all you want is access to a long long type! MacDonald said that doesn't address portability for exact types. Gwyn argued that exact sizes don't guarantee portable layout; least size types are more

useful, and he thought we had decided exact size types weren't useful. MacDonald and Farance both replied that we go through this at every meeting and end up confirming that we need 3 types: exact, at least, and fastest.

There was some discussion of extending the specification for supporting 24, 30, 60, or 80 bit machines.

Farance noted that applications know the sizes they need and we shouldn't try to "fix" them. His proposal will solve all these problems. Plum noted, however, that <inttypes.h> can solve problems right now. Kwan added that <inttypes.h> is not intended to solve all portability problems.

Plum brought us back to the issue at hand, how to mange I/O. We seemed to be leaning towards macros rather than extensions.

- Conversion functions
  - Only 2 conversion functions are necessary
    extern intmax_t strtoimax();
    extern uintmax_t strtoumax();

- Constants
  - #define __CONCAT__(A<B) A ## B
  - #define INT16_C(c) (int16_t) c
  - #define UINT16_C(c) ((uint16_t) __CONCAT__(c,u))
  - #define INTMAX_C(c) ((int64_t) __CONCAT__(c,ll))
    implementation defined

- Some issues
  - Should the extended type be promoted to "int" when used as a parameter to functions without prototype?
    It loses its meaning if promoted
  - In general, the following types are not always compatible:
    int32_t
    the base type int
  - The library used must be consistent with the application in using the same header.
  - Integral promotion rules should be the same as the underlining base type

Gwyn believes promotion will "fall out" depending on type used as target.

Plum repeated that we need to resolve all remaining questions so that users can start using <inttypes.h>!

Farance presented an alternative approach, detailed in N355.

- Purpose
  - Applications with known (portable) precision
    - Need more types and long long is not enough
    - Need more precision or certain level of precision
  - Applications access data with known representation - portable data
    - values representable

This generates two intermediate level problems:

(1) Providing known precision to applications without constraint of implementation-defined types (int, long, short, char).
(2) Providing known data structures that may be ported to different machine architectures (e.g., via FTP or NFS). This issue concerns bit/byte ordering and bit/byte alignment.

- Integer precision
  - need:
    - fastest with >= n bits
    - smallest with >= n bits
    - exactly n bits
  - proposed syntax:

```
long:n      /* fastest */
short:n     /* smallest */
int:n       /* exact */
```

If you don't have these, user must manage, and this is error prone.

- Bit/Byte ordering
  - Write on one machine, read on another
  - FTP and NFS are other examples
  - proposed syntax:
    - bit order

```
msb      /* msb is bit 0 */
lsb      /* lsb is bit 0 */
```

Used for bit fields in structures

    - byte order

```
bigend
littleend
```

    - example:

```
lsb bigend int:32
```

- Without this, users have to make mappings for these orderings, usually constructing a table with offset, bit order, byte order information.

MacDonald asked if bit ordering was the same as left pack vs. right pack. Gwyn believes the bit ordering seemed to apply to the overall struct rather than just individual bit fields. Anticipates problems if both were specified in the same struct. Farance clarified that it indicates whether to fill on left or on right.

Plum raised two points: (1) there are more than 2 byte orderings - at least 3 or 4; and (2) if we consider this direction, ought to consider architectural concerns. Will we be biasing application programs towards inefficient solutions? Farance believes there are also maintenance aspects to consider in addition to efficiency.

Gwyn thinks this could potentially prevent a lot of bugs in network programming.

- Bit/Btye alignment - bit fields
  - suggested syntax:

```
bitalign: 0      /* force next bit */
bitalign:N       /* round up to N bits */
bytealign:0      /* force next byte */
```

```
bytealign:N     /* round up to N bytes */
```

Note: bytealign:0 is similar to "pack" on some systems

- example:

```
{ bytealign:0;
  lsb bigend int:32 A;
  bytealign:4;
  lsb bigend int:32 B;
}
```

Note: you can take the & of fields A and B now!

Mooney asked if `bytealign` affects every field following. No, only the next immediate field.

Jaeschke asked what is the type of `&` of one of these bit fields, and how do I declare it? Just `int:32 *`. Jones asked what if you pack into the wrong "end" for addressing on machine and then take `&`. Could get around this with `char *`.

- Open issues
  - printf/scanf - would go in same direction as Kwan's proposal.
  - need a lg2 macro - to derive # bits from range

- Comparison to Kwan's proposal? Bottom line is that this approach is easier.

SV In favor of Kwan continuing with <inttypes.h> proposal?
17 Yes. 0 No. 0 Abstaining.

In favor of including such a facility as Farance outlines in the TR?
4 Yes. 4 No. 10 Abstaining.

MacDonald expressed doubts about it being ready for the TR. Jaeschke said it could at least go into rationale if not ready.

Coonen suggested that maybe the solution should proceed more along the lines of filters (library support) rather than compiler support.

Kwan stated that he would have no problem with this as an extension, but would not want it as part of next standard because it adds too much complexity.

Farance asked if ok to present at the next meeting. Jaeschke indicated there may be a problem getting agenda time since will need to give preference to endorsed proposals, but there would be no problem with a paper for the next meeting.

Plum indicated the motivating factor would be finding a vendor interested in implementing this approach. Gwyn observed that none of the vendors present expressed such an interest.

9. DPCE and Array Syntax [N329/94-014, N330/94-015, N336/94-021, N340/94-025]

Farance announced his resignation as focal point for the DPCE:

"I am leaving DPCE and resigning as chair. I've spent 5 years representing X3J11's interests in this subcommittee. I believe DPCE does not want to follow the will of X3J11. This is my reason for resignation. I've tried focusing the DPCE subcommittee on the standards process and X3J11's needs. Many times, DPCE members have said they are fully aware that X3J11 won't approve the DPCE proposal. I can't, in good conscience, continue with a subcommittee that continues not to respond to X3J11. As I part, I'd like to leave with some constructive advice to help X3J11 and DPCE: (1) X3J11 and DPCE must keep track of action items. DPCE should fulfill them to the best possible. (2) Separate the secretary and technical editor's positions. In X3J11 with Plauger and Rosler, they provided a system of checks and balances. Right now, DPCE has no control (as in correlation) of transactions in the meeting as they correspond to document changes. (3) DPCE should realize that once the public review starts, the document will be out of their hands. They should establish procedures so that disasters, like Hansberry, don't happen. Finally, I wish Dave Keaton good luck and I will help in whatever way to assist Dave as chair."

Keaton responded that he believes DPCE is 'fully aware that X3J11 won't approve the DPCE proposal' as an amendment to the C language, but the technical report is a different matter. He believes the discrepancy between this and Farance's statement is just a matter of remembering things differently.

Keaton reported on the overall status of the DPCE subgroup:

- State of DPCE proposal.
  Document has doubled in size since December.
  Specification is more complete.
  A fair amount of edits remain, marked in the document as <<notes>>.
- Summary of changes since last meeting
  A list of promised changes appeared in the X3J11.1 minutes. They were completed.
- Review of X3J11 action items
  More detail will be given in the technical editor's report.
  Farance will report on the F90 VLA/Data Parallel action item separately.
  Meissner had an action item to find the source for definitions of terms not found in the C standard, but he is not attending this meeting.
  Actionable questions have been discussed.
  Separating layout & context from shape (March & e-mail).
  Left indexing (e-mail).
  Votes at March DPCE meeting.
  Separate layout from shape.
  1 yes, 3 no, 1 not voting.
  Separate context from shape.
  1 yes, 4 no, 0 not voting.
- Procedures for public review
  E-mail, ftp, and X3J11 discussion.
- Schedule/time-line for completion of DPCE proposal.
  Goal is to complete the document by the September DPCE meeting, for inclusion in the following X3J11 mailing and the NCEG technical report.
  Aggressive schedule including about 62 items to be edited.
  About 1/3 to be finished by July 1, the rest before the DPCE meeting in September.
  Almost 1 edit item per working day.

September meeting to provide final reviews and corrections before second mailing.

Keaton announced there will be evening meetings for DPCE, Monday and Wednesday, 7-9PM.

Stanberry gave the technical editor's summary of changes to the document.

- Major changes in the document include:
  - shape type, shape variable, "same shape" specifications
  - nodal functions
  - parallel pointers
  - library functions

- Other major writing/refining occurred in the following areas:
  - array subscripting (3.3.2.1)
  - address and indirection operators (3.3.3.2)
  - parallel indexing (3.3.3.5)
  - additive operators (3.3.6)
  - elemental functions (3.3.2.2, 3.5.3, 3.7.1)
  - layout specification (3.5.2.4)
  - parallel object declarators (3.5.4.4)
  - contextualization (3.6.7)

- Miscellaneous editorial changes occurred everywhere, adding <<notes>> where inconsistencies or incomplete specifications were identified, adding meta-words, adding examples, adding outline for Appendix A, adding an index, and inserting numerous typos.

- Major revisions were made to the indicated sections of the document for these changes:
  - shapes
    - 3.1.2.6 - made "compatible shape" and "composite shape" specification consistent with the 3 kinds of shape types (fully specified, partially specified, fully unspecified)
    - 3.2.3 - revised description of operations on parallel operands to require "same shape" determined as structural, rather than name, equivalence.
    - 3.3.16.1 - assignment of shape variables revised to use copy rather than aliasing semantics
  - nodal functions
    - 3.3.2.2 - added constraints and semantics of function calls for nodal functions
    - 3.5.3 - added nodal type qualifier
    - 3.6.6.4 - added semantics for return statement for nodal functions
    - 3.7.1 - added constraints and semantics for function definitions for nodal functions
  - parallel pointers
    - 3.3.3.2 - added constraints and semantics for & operator applied to produce a parallel pointer, and corresponding semantics of * operator applied to a parallel pointer
    - 3.5.4.1 - revised constraints and semantics of pointer declarations for parallel pointers
  - library functions

- 4.1.2 - added <dpce.h>
- 4.14 - added descriptions of dpce library functions, most of which are elemental, allowing them to be used on parallel or non-parallel arguments.

- Nodal functions
  - An invocation of a nodal function occurs as if the function is invoked once on each node of the execution environment in SPMD style; that is, as if a separate thread is spawned on each node to execute the function body. Nodal functions therefore provide an escape to a multi-threaded programming model. These threads, one per node, are only required by the execution model to synchronize on return from the nodal function.
  - On each node the body of the nodal function executes in a temporarily established single-node environment. That is, during the execution of a nodal function, a call to **positionsof(physical)** will return 1. The execution environment of the caller is re-established upon return from the nodal function.
  - Example

```
shape [10] physical;   /* predefined shape */

nodal int fun(int:void a)
{
   int sum = 0;

   sum += a;
   return sum;
}

...
shape [100 block (10)] S;
int:S   x;
int:physical result;

result = fun(x);
...
```

Executes function body "as if":

```
shape [1] physical;
shape [10 block(10)] S;
```

Returns sum of elements on each node; each function invocation contributes one element of parallel result, which has **physical** shape.

MacDonald asked several questions. Where is layout specification? It is now in the document (see §3.5.2.4). Note that the rankof(physical) == # of processors. What are the restrictions on nodal functions? A nodal function may call only nodal and elemental functions. I/O forbidden? Yes. What about debugging? Unresolved issue. Do nodal functions support private, global variables? Cannot reference any identifier with file scope. (See §3.7.1)

There was discussion (Tydeman, Kwan, Thomas) about whether or not sum needs to be initialized to 0 in the example, and the semantics of the += binary operator

as opposed to the semantics of the += unary operator. Stanberry replied (erroneously) that no initialization was required; Alpern pointed out this error afterwards, and the example was corrected in these minutes to reflect this. The concern was that the semantics of binary += was misleading if no initialization was required, and inconsistent with ordinary C use. The semantics of binary += is intended to be exactly as for C non-parallel operands; the technical editor apologizes for this error in both example and discussion.

- Parallel pointers - & and * operators
  - Constraints

    If the operand of the unary & operator is a parallel-indexed expression, at least one of the index expressions shall be parallel. If the operand of the unary & operator is a parallel-indexed expression, the shape of the parallel index expression(s) shall be the same as the shape of the expression being indexed.

    Note: you cannot take the address of an element of a parallel operand.
  - Semantics

    The application of & to a parallel-indexed lvalue produces a parallel pointer whose shape is the shape of the parallel index to that parallel lvalue. If the operand has type "parallel T of shape S parallel-indexed by expressions of shape S," the result has type "parallel pointer of shape S to parallel T of shape S."

    The application of & to a parallel lvalue produces a pointer to that lvalue. If the operand has type "parallel T of shape S" the result has type "pointer to parallel T of shape S."

    The result of the * operator applied to a pointer to parallel pointer of shape S to parallel type T also of shape S is a parallel lvalue of parallel type T and shape S.

    The result of the * operator applied to a pointer to parallel type T of shape S is a parallel lvalue of parallel type T and of shape S.
  - Examples

```
float a, b;
shape [10]S;

int:S x;              /* parallel int */
double:S y,z;         /* parallel double */
int:S *p;             /* pointer to parallel int */
float *pp2f:S;        /* parallel pointer to float */

p = &x;          /* Assigns p to point to parallel
                    int x */
pp2f = &a;       /* Assigns each element of pp2f to
                    point to float a */
[3]pp2f = &b;    /* Assigns 3rd element of pp2f to
                    point to float b */
pp2pd = &y;      /* Assigns each element of pp2pd to
                    point to parallel double y */
[2]pp2pd = &z;   /* Assigns 2nd element of pp2pd to
                    point to parallel double z */
```

88

```
[7]pp2f = &b;   /* Assigns 7th element of pp2f to
                   point to float b */
*p;             /* Denotes the parallel int x */
*pp2f;          /* Denotes a parallel float of shape
                   S; in this case equal to a in all
                   positions except equal to b in
                   positions 3 and 7 */
```

Farance asked about whether parallel pointers are global pointers? Can you access any area, any cpu, with a parallel pointer? Depending on the definition of global pointer, could be yes or no. Whether, for example, a parallel pointer is wider than `void *` is an open issue in DPCE. Also, parallel pointer is for entire object, not for individual elements. Farance outlined example of where one needs global pointers, as in sorting large data–too much to write to file, so use global pointer, but you need to be able to access any cpu. MacDonald noted that you could always write indices to file. Alpern suggested you could do this with a nodal function.

MacDonald asked about contextualization–is this new? No. Does this mean that every operation is performed under context? Yes. MacDonald did not like this as it adds overhead to all operations.

What is the definition of axis? DPCE wants to rely on standard definitions for such terms, rather than define them in their document. Meissner was going to investigate what the official dictionary is for such terms, but this is still an open issue.

Gwyn asked if the model assumes peer nodes; should also consider hierarchical or cluster architectures.

Thomas asked if Plum had been consulted on C++ liaison issues. Stanberry and Keaton affirmed that he had attended the Monday evening session to provide such information.

Keaton reported on the March DPCE meeting.

- Separation of layout from shape
  - Pro's
    - Conceptually cleaner, elegance of expressiblity
    - Allows operations on objects of different layouts
    - No significant performance cost on non-distributed memory
  - Con's
    - Disallows optimizations knowing objects are the same layout–significant performance hit
    - Transparency of performance lost
    - Must be coupled conceptually on distributed memory architectures anyway

Farance stated that making visible to the programmer was a quality of implementation issue. He also reports that no dynamic layout supported as in HPF.

- Separation of context from shape
  - Pro's
    - Conceptually separate: storage vs execution time concepts
  - Con's
    - Artificially separates concepts always used together

Complicates explanation of what is affected by context

Farance argued that should look at uses without shape. Should not be tied to shape since feature of execution not data.

Alpern commented that it it possible to lexically determine context as opposed to dynamically storing context with shape. MacDonald added that HPF uses lexical determination of context.

Keaton then reported on discussion on the DPCE email reflector on parallel (left) indexing alternatives. Farance suggested that something should indicate in data declaration which indices are parallel; not necessary to use syntax to distinguish parallel from non-parallel indices. Keaton reiterated the DPCE position has been to use syntax to indicate potential communication.

Farance objected that this discussion of alternatives bundles 3 concepts: ALO's, parallelism, and C++ compatible alternatives.

MacDonald asked for clarification of parallel indices alternatives–are they still restricted to be the outer/left ones? Currently, yes, but would want to leave open for future extensions.

Alpern expressed concern about so-called C++ compatible alternatives. It is hazardous to make it look like C++ if semantics are quite different.

There was some discussion of the motivations for the different alternatives, and then Keaton conducted a preference poll (can vote for more than one).

SV Preferred alternative:

```
3     (1)  [i][j]A[k][l]
1     (2)  A[:i][:j][k][l]
1     (3)  A(i)(j)[k][l]
9     (4)  A(i,j)[k][l]
5     (5)  A[i][j][k][l]
```

MacDonald presented N336, adding shape to iterators.

- Brief review of iterators:
  - Philosophy: DPCE is single-threaded (except for nodal functions). Iterators allow multi-threading.
  - Ordered and unordered iterators

MacDonald noted that Analog Devices has added iterators to the GNU C compiler. He also admitted that iterators would not be amenable to a C++ class implementation.

- Shapes + Iterators–Goals: to be able to describe data layout
  - Useful programming model for distributed memory machines
  - Small extension to the language
    - Easy to implement syntax
    - Easy to learn
  - Attractive even on shared memory machines

- Shape

- New type specifier 'shape' is used to declare objects that record extent and (optional) layout information.

```
shape S[100][200];
shape S1[100 block(10)][200 block(20)];
shape S2[100 scale(1)][100 scale(2)];
```

- Uses of shapes
  - In a new form of array derivation.

```
shape S[100][200];
double a[S], b[S];
```

  - In declarations of iterators.

```
unord i = S, j = S;
```

  - In function calls.

```
f(S,a,b)
```

- Shaped iterators
  - Shaped iterators are used for operations:
    (1) with a common shape for all shaped operands, and
    (2) that do not require communication.

```
shape S[100][100];
double a[S], b[S];
unord i = S, j = 100, k = 100;

a[i] = b[i] + 1.0;  /* no communication */
a[j][k] = b[k][j];  /* communication */
a[j][k] = b[j][(k+1)%100];  /* communication */
```

- Function calls
  - Shaped function parameters are passed by reference.

```
void f(shape S, double a[S]) {
   unord int i = S;
   a[i] += 1.0;
}
```

  - An alternative form uses explicit pointers, and requires all references to a to use shaped iterators.

```
void f(shape S, double *a) {
   unord int i = S;
   a[i] += 1.0;
}
```

- Slices
  - Another new keyword:

```
shape S[100][200];
```

```
slice S[0;50;2][0;100;2]S1;
```

- Reference a portion of a shaped object.

```
double a[S];
unord int i = S1; /* both coords even */

a[i] = 0.0;
```

- Align a smaller object with a portion of a larger object.

```
double b[S1];

b[i] = a[i];
```

- Like shapes, slices and sliced objects can be passed as function parameters.

```
shape S[100][200];
slice S[0;2;99][;]S1;
slice S[;][0;2;199]S2;
double m[S];

void f(slice SL, double a[SL]) {
  unord int i = SL;
    a[i] = 0.0;
}

/* Set boundary of m to zero */
f(S1,m);
f(S2,m);
```

- Summary
  - Shapes, slices, and iterators could give a model with:
    - explicit expression of layout and parallel operations
    - minimal new syntax
    - no implied temporaries for intermediate aggregate results

Gwyn asked for clarification of whether a type specifier was needed for **iter** and **unord**; examples use both with and without **int**.

Alpern asked what kind of things can be done with A[S] with respect to pointer arithmetic. Can access individual elements (A[2]) but can't take address of an element (&A[2]). So does this mean that A[2] no longer means `*A + 2 * sizeof(element_type)`? Mapping takes place to right element on that processor.

Gwyn asked if dynamic slices were allowed. Yes, desirable.

Farance presented his array slicing proposal, prepared as an extension for DPCE, N340.

- DPCE Array Slicing – Rationale
  - Problem to solve:

- Supplying Fortran-like triplet notation
- scatter-gather

- Overall problem: DPCE notation of shape doesn't allow interaction of anonymous shapes

```
shape [10]S;
shape [10]T;

int:S C;
int:T D;
int:T E;
int:[10] F;
int:[10] G;

D+E         <- OK
C+D         <- ERROR-incompatible because derived
                 from different shapes
F+G         <- ERROR-incompatible because derived
                 from different shapes
```

- Proposal:

```
shape [10]S;
int:S A;
int I;

I = += [0:2]A * [3:5]A;
```

- Even if we relax shape derivation, still have problem

```
shape [3]U;
shape [3]V;
int:U A;
int:V B;
int I;

[1]A = 0;
where (A != 0)
{
   I = += B/A;
}
```

- Rule #1 shape compatibility is dependent upon the "name" of the shape it is derived from, not the "value" of the shape (i.e., its rank + dimensions).
- Rule #2 shapes must be explicitly declared (i.e., a name created) to facilitate shape compatibility.
- Rule #3 context is tied to shape and, therefore, associated with the storage of the shape and all derived objects.

- Layout is unrelated to shape but tied to shape in DPCE. In F90 and HPF, shape derivation and layout compatibility are not required.

- Summary:
  - DPCE slicing can only apply to left indexes, not right indexes.

- Shape/type compatibility requires more than rank+dimensions.
- Shape must be declared, i.e., create a name for derivation.
- Context is tied to shape, cannot be separated.
- Layout is unrelated to slicing semantics.

Stanberry pointed out that this is an erronenous characterization of DPCE proposal: these arguments are flawed by wrong assumptions.

- DPCE slicing
  - C-style triplet
    ```
    [[first; length; stride]]
    [[first; length]]
    ```
  - Fortran triplet
    ```
    [[first : last : stride]]
    [[first : last]]
    ```
  - scatter/gather
    ```
    [[index]]
    ```
  - all
    ```
    [[]]
    ```

- VLA extensions
  - C-style triplet
    ```
    [first; length; stride]
    [first; length]
    ```
  - Fortran triplet
    ```
    [first : last : stride]
    [first : last]
    ```
  - scatter/gather
    ```
    [index]
    ```
  - all
    ```
    []
    ```

- Can't get a smaller ALO than what you started with because of gather/scatter capability with respect to shape.

```
shape [10][20] T;
int:T B;

[.][1]B <- still is shape T
```

Stanberry pointed out that, while this example is correct, the general statement is another erroneous characterization of the DPCE proposal.

Farance presented a summary of parallel mechanisms, which he will also post to the email reflector:

Type	Same	Different	Scope
scalar/array promotion	Elements	None	Expression
data parallel	Statement	Context Mask	Statement
iterator	Expression	unord	Statement
for	Statements	if-else, switch	Block
X3H5 pfor	Block	None	Block
context	Statements, Subroutine	where	Subroutine

elemental function	Subroutine	None	Subroutine
nodal function	None	Subroutine	Subroutine
POSIX threads	Subroutine, Process	Thread ID	Process
fork/exec	Process	exec	Multiprocess System

He would like feedback on this summary.

Farance gave a summary of APL/VLA, DPCE Slicing, and Parallelism:

- Concepts may be separately defined
  - shape – varying size arrays
  - alo/carrays – default stand alone behavior
  - layout – distributed memory
    - near, far, huge ptrs
  - ptr/value, shape, layout matrix of arg passing combo
  - selectors: index, slice, scatter-gather
  - parallel mechanisms:
    - scalar/array promotion
    - iterators
    - where statement – not slice
    - X3H5 – not sure
    - shapeis(?) – similar to elemental

- Can get same (or better) functionality and performance as DPCE

10. Defect Reports [N347/94-032]

The committee divided into 5 working groups to review the current defect reports, led by Jones, Gwyn, Zeeb, Ziebel, and Kwan.

The following guidelines were suggested by the officers for procedures at this meeting:

- Many already have responses adopted at the Kona meeting. If anyone has objection to one of these, take it to the leader of that group. We don't want to reconsider DR's to which no one has objection to the suggested response.

- "The standard doesn't clearly answer this question" is a legitimate response if after one meeting's consideration, can't decide among ourselves. Meanwhile, there is no portable solution. Best addressed as part of the revision of the standard. Can always reconsider at subsequent meeting if new insight is discovered.

Gwyn asked for clarification on this: if it is not specified in the standard, don't we have to supply a fix to the standard? Also, we don't want to respond, "this takes too long to determine." Plum agreed, this is a guideline only. He reiterated that we don't want to drop such items from the standard revision.

Gwyn asked that typos also be flagged by the subgroups and reported directly to him. Also, all wording for responses (new or changes to suggested ones) should

be submitted to him by email by the group leaders. Gwyn will also provide minor editing of responses for conformance to format of the RR and TC as appropriate.

Note: any defect report NOT explicitly listed here was not discussed in full committee, with the understanding that the suggested response, if any, in N347/94-032 will be used. The presenter for each DR is indicated in ()'s. For brevity, the questions are not repeated here.

#60 - (Jones) Agreed with the suggested response.

#61 - (Gwyn) Agreed with the suggested response.

#62 - (Zeeb) Agreed with the suggested response.

#63 - (Ziebell) Use Plauger's response as "discussion" and as response say:

> The standard imposes no requirement on the accuracy of floating point arithmetic.

#64 - (MacDonald) Agreed with the suggested response.

#65 - (Jones) Basically agreed with the suggested correction, but add Feather's rationale:

> The committee affirmed that the intent of this wording is that a program (such as that above) whose output varies only according to the locale selected, and does not rely on the presence of a specific locale other than the C locale or that selected by "", was always intended to be strictly-conforming. Nevertheless, it is agreed that a strict reading of the cited extract from subclause 7.4.1.1 could be read as making such programs depend on implementation-defined behavior.
>
> The committee reaffirms that programs that depend on the identity of the available locales, as opposed to their contents, are not strictly conforming.
>
> The committee believes that the term "implementation-defined" in the first sentence of the extract from 7.4.1.1 was intended in the sense of "implementation documented". However, they were loath to introduce a new term with possibly new conformance requirements, in a Technical Corrigendum. The committee noted that the term "locale-specific", while making the sentence read somewhat awkwardly, carries the necessary requirements (the implementation must document the relevant details).
>
> The committee also decided that, though the question only addresses one issue to do with locales, that the above discussion applied to all instances where the behaviour of an implementation depends on the locale. For this reason they decided to address all such issues at this time.

#66 - (Gwyn) Agreed with the suggested response rather than the suggested correction. Believe it is unambiguously stated in the standard.

#67 - (Zeeb) Agreed partially with the suggested response, but edit to split answers to (c) and (d); last paragraph of suggested response really answers (d) and (e).

Plum asked, with respect to question (b), what is a version of a type? Suggested just to use reference to 6.1.2.5. Further suggested that (d) was beyond the scope of the standard.

There was discussion of when new types could not be used–how could a standard conforming program recognize them? Everyone agreed the answer to (e) was no.

Mooney asked if we want to extend answers for enumeration types. No, don't go beyond what was asked.

Final suggested response:

(a) <same as suggested response>
(b) Yes, see 6.1.2.5.
(c) <same as suggested response before for c and d>
(d) Beyond the scope of the standard.
(e) No.

#68 - (Ziebell) Suggested that the committee's original response be selected, except to change (a) to just say "Yes."

#69 - (MacDonald) This multi-part question from Feather makes one ponder what exactly does he have in mind when asking these questions? BCD?

Plum gave a brief history of this part of the C standard specification. The intent was to disallow BCD and certain other architectures. Really a conscious decision to endorse only one's and two's complement and signed magnitude architectures. C was not designed axiomatically but with experience on specific architectures.

There was considerable discussion (Gwyn, Farance, MacDonald, Kwan, Keaton):

• State underlying principles of intent as preamble. Never our intent to completely characterize the behavior of the underlying binary systems.
• We understood some architectures would have bits not used in int representations.
• Unsigned arithmetic intended to be portable and to be able to have all bits in value examinable.

Where the standard is not specific, may not be able to give answers to his questions.

After a couple of iterations, decided on the following responses:

(a) Footnotes are not normative and the legality of a footnote is beyond the scope of WG14/X3J11.
(b) Yes, the footnote is correct.
(c) No, no such requirement is known.
(d) No.
(e) If by the term "bit pattern" you mean values, the answer is Yes. If by "bit pattern" you mean representations, the answer is No.
(f) No.
(g) Not applicable since it is unclear what is meant by "bit pattern" and "value" in the question.

(h) Yes, provided there is no other violation of the standard.
(i) Yes, provided there is no other violation of the standard.
(j) No, it's not a pure binary system.
(k) Yes, provided there is no other violation of the standard.
(l) Yes, provided there is no other violation of the standard.
(m) Yes, because subclause 6.1.2.5 states that the representations of positive signed integers have the same representations as the corresponding unsigned integers, and because signed integers use a pure binary numeration system. The committee intended to permit one's complement, two's complement, and signed magnitude implementations.
(n) No.
(p) Yes, because subclause 6.1.2.5 requires unsigned integers to behave as if a result "is reduced modulo the number that is one greater than the largest value that can be represented," and unsigned integers use a pure binary numeration system.
(q) No, the memory occupied by a value of an integer type is allowed to exceed the number of binary digits used to represent the actual value.
(r) Yes, same reason as question m.
(s) No, same reason as question q.
(t) Not applicable.
(u) Yes, the expressions must evaluate to 0 or -1.

#70 - (Jones) May a strictly conforming program take advantage of "same representation" requirements? Agreed with Plauger, can't take advantage of same representation requirements. Suggested response:

> The program is not strictly conforming since many pre-existing programs assume that objects with the same representation are interchangeable in these contexts, the standard encourages implementors to allow such code to work, but does not require it.

Gwyn recalls that the committee did intend for int argument punning for old style functions to be strictly conforming–stronger than just "allowed." Others agreed.

Plum stated that unless we can prove the words actually state the requirement, the bottom line is the same. Noted that the same rules are in C++, but even with its stricter checking, can still produce code with undefined behavior.

MacDonald and Gwyn believe that function call semantics imply that incompatible types after promotion of arguments results in undefined behavior.

Plum asked about examination under union: not all representation-dependent uses are undefined, some are implementation defined.

#71 - (Plum) Accept the suggested response from Feather, striking the editorial insert from Plauger and following sentence ("[not true, just int - pjp]" and "A TC is required.").

Plum noted that C++ is going to allow long and even unsigned enumerations.

Jones and Guilmette argued that an explicit change to 6.5.2.2 is required. Accept the suggested correction to 6.5.2.2, but reword the last sentence to read "..., but shall be capable of representing all the members of the enumeration."

#72 - (Zeeb) Agreed with the suggested response.

#73 - (Ziebell) Suggested that we accept the Kona response, but there was discussion .

Guilmette thought we needed to clarify what is meant by "The dot selection operator is at liberty to require the complete struct denoted by its left hand side to be accessed." Gwyn noted that "." doesn't imply there is or isn't an access. Keaton suggested that this sentence be removed. Gwyn suggested it be prefixed with "In these cases, ..." Jones stated that in 6.3.2.3 the semantics requires a member of a struct or union object and there is no such object, so the behavior is undefined. Plum suggested we go with the wording as is.

SV In favor of response as written to A, B, C of DR#73?
10 Yes. 2 No. 3 Don't know/don't care.

Guilmette also questioned whether responses F, G, and H are really the same since they involve `&`. Others believe they require the same analysis.

#74 - (Kwan) Agree with all but part (d) of the suggested response. The alignment of a struct need not be the least common multiple of the alignment of its members. So the answer to (d) is Yes.

Gwyn noted that the answer to (c) should restate the answer with which they are agreeing!

#75 - (Jones) Must malloc result be aligned for any type or just types that fit? Edit the suggested response as follows:

> Subclause 7.10.3 requires allocated memory to be suitably aligned for <u>any</u> type, so they must compare equal.

#76 - (Plum) Endorse the suggested response, changing `a(10)` to `a[10]`.

#77 - (Zeeb) Discussion of whether suggested response actually answers the questions. Believe the answers are "yes" but the response doesn't say that. There was more discussion, furiously agreeing. Rewording suggested was:

> The standard does not explicitly state that the address of an object remains constant throughout the life of the object. That this is the intent of the committee can be inferred from the fact that an address constant is considered to be a constant expression. The framers of the standard considered that it was so obvious that addresses should remain constant that they neglected to craft words to that effect.

#78 - (Ziebell) Recommend rewording of part c of suggested response to "Yes, `h` can return `0`."

Plum as C++ liaison made an appeal not to rely heavily on "different address" meaning "different function" because of inlined functions since the linkage of inlined functions has been changed to extern. Taking the address of an inlined function implies generated code for a thunk.

#79 - (Guilmette) Revised suggested response as follows:

(a) agree with response except delete 2nd sentence.
(b) The standard library functions must have external linkage (see subclause 7.1.2.1, bullet 3).
(c) agree with suggested response.

#80 - (Jones) String literal merging. Need to clarify or qualify the meanings of "string literal" which means different things in different phases of translation.

MacDonald and Gwyn believe we want to allow the sharing of identical string literals since they can't be changed. Gwyn stated that it follows that we also intended to allow sharing of tails of string literals. Jones asked if we want to explicitly say that we allow tail merging.

SV In favor of allowing tail merging.
14 Yes. 1 No. 2 Don't know/don't care.

SV Believe intent was to allow tail merging.
3 Yes. 0 No. Lots don't know.

Suggested response:

When the last paragraph of 6.1.4 refers to "string literals" it is referring to the static arrays created in translation phase 7 as specified in the previous paragraph. Although the existing wording of the standard may imply that only completely identical arrays need not be distinct, this was not the committee's intent.

Suggested correction to last paragraph of 6.1.4:

These arrays need not be distinct provided their elements have the correct values; if the program attempts to modify such an array, the behavior is undefined.

#81 - (Plum) Accept the suggested response.

#82 - (Zeeb) Agreed with the general thrust of the suggested response.

Gwyn and Jones discussed need to make correction to the standard to enforce intended pairing of calls to **va_start** and **va_end** macros. Need to revise (a) and add suggested correction.

Edited suggested response:

(a) All functions listed except for `f3` contain strictly conforming code. The function `f3` violates the intended requirement for **va_start** and **va_end** to be invoked in matching pairs, as reflected in the following correction.
(b) There is nothing described in this section that would make such an implementation non-conforming.
(c) No.

Suggested correction to 7.8.1, end of paragraph:

> The **va_start** and **va_end** macros shall be invoked in corresponding pairs in the function accepting a varying number of arguments, if access to the varying arguments is desired.

Suggested correction to 7.8.1.1, end of second paragraph:

> **va_start** shall not be invoked again for the same **ap** without an intervening invocation of **va_end** for the same **ap**.

#83 - (Keaton) Use suggested correction by Feather and Plauger.

#84 - (MacDonald) Need to decide if parameters in prototypes are objects. If so, can't be incomplete types by 6.1.2.2 and 6.5. See also DR#104.

Plum believes underlying intuition is that there is an object but that it isn't needed until function call. MacDonald suggested then our response should be "This is undefined behavior so not strictly conforming" but not a constraint violation. Gwyn says this is ok for DR#84 but not for DR#104. Guilmette believes lots of code think this is standard conforming since this is obscure. Plum suggested we consider improving this area for the next revision of the standard. Keaton pointed out that this would make DR#103 undefined. Jones agress that we need to consider all these together.

So this is an open issue: DR#103 and DR#104 responses depend on our response to this DR.

#85 - (Jones) Agreed with the suggested correction from the Kona meeting, rather than the Plauger response.

#87 - (Zeeb) This DR was related to DR #117, and Plauger had attempted to use the same response for both, but it was generally agreed that each needed its own answer.

Plum reported that the C++ proposal for sequence points will be C with some exceptions, and is also more detailed than C. For example, (1) C doesn't say function calls can be interleaved; (2) the "undefined behavior if same object is modified twice" not intended to disallow modifying more than one struct member; (3) more precise: if any ordering of all possible orderings of operations would violate the "modified twice" rule, then the program is not strictly conforming.

MacDonald doesn't want to preclude parallel execution; remove thread references from the response.

Plum noted that for A and B, the behavior is not undefined, but it is unspecified whether `g` will be `1` or `2`; however, since those are the only choices, it is not indeterminate.

SV For A and B in the examples of DR#87, believe the behavior is unspecified but not undefined.
3 Yes. 0 No. 14 Not sure.

After more word crafting, replaced Plauger's suggested response with:

> In lines A and B, the expressions do not exhibit undefined behavior, but because the order of evaluation of the operands of the addition operator are unspecified, it is unspecified whether g will attain the value 1 or 2. Line C violates the quoted restriction from subclause 6.3, so the behavior is undefined.

#88 - (Ziebell) In this DR, 35 separate puzzles are given. Suggests the committee response should be to ask the submitter (Feather) to be more explicit in what defect is being reported. Suggested response:

> We request that you state the defect more directly.

#89 - (Guilmette) Agree with Feather's suggested correction, but add "that" to end of text to be replaced.

MacDonald pointed out that this change makes a diagnostic requirement where none existed before. Jones stated then it shouldn't have been in the constraints section anyway.

#90 - (Jones) Insert clarifying preface wording to the suggested response:

> The first call contains no locale specific characters and must produce the obvious output. The remainder of this response addresses the subsequent calls.

#92 - (Zeeb) Replace the suggested response with reference to DR#60.

#93 - (Keaton) Use suggested correction and add Feather's E-mail as discussion:

> The committee observed that conforming freestanding implementations tend to vary widely in the library facilities provided, and that the simple binary choice implied by the above text is really a continuum. It was also noted that it is difficult to provide a C implementation with no reserved names (not even those beginning with two underscores). It was therefore felt unreasonable to restrict the names available to implementers of freestanding implementations compared with hosted implementations.
>
> The committee notes that certain freestanding programs (such as Unix kernels) have tended to use names such as "exit", but it was agreed that existing practice means that the authors of such programs must already be prepared to change such names when using certain compilers.

Gwyn asked to clarify if this would allow a user to access library functions. Yes!

#94 - (MacDonald) The subgroup looking at this disagreed with the suggested response, and preferred Guilmette's suggested correction.

SV In favor of adding new constraint to require the type of an expression in a return statement to be compatible with the function return type.
8 Yes. 8 No. 0 Don't know/don't care.

Jones noted that we would have to revise all the other DR's that we answered with "as if by assignment." MacDonald disagreed since this just makes return statements symmetric with constraints on argument types in function calls.

Since the committee was evenly split on this, the suggested response (status quo) was accepted.

#95 - (Jones) Accept the suggested response, but remove initial sentence fragment from second paragraph.

#97 - (Zeeb) Replace both suggested responses with:

> This was answered in DR#40, question 6. This code is not strictly conforming.

#98 - (Keaton) Use the suggested response.

#100 - (Jones) Accept suggested response.

#102 - (Zeeb) Agreed with the suggested response, but at Guilmette's suggestion, also add "See response to DR#17 question 3."

#103 - (Keaton) Suggested edited form of Plauger's response:

> The types of the parameters are rewritten, as in subclause 6.7.1. No incomplete object types are involved.

Guilmette still questions whether a TC is required to correct 6.5. Gwyn and Keaton argued that it is sufficiently explained in 6.7.1 and the standard must be read as a whole. MacDonald noted that 6.7.1 refers to function definitions and this is a function declaration. Need to look at DR#47. Left as an open issue–see DR#84 and DR#104.

#104 - (MacDonald) Open issue–see DR#84 and DR#103.

#105 - (Jones) Change response to:

> This error was corrected in response to DR#017, question 3.

#106 – (Jones) Jones will collect comments on this to try to formulate a response.

#107 - (Zeeb) Accept the suggested response, but add to (b):

> Furthermore, if NDEBUG is a macro, `assert` is defined as
>
> ```
> #define assert(ingore) ((void)0)
> ```
>
> With this definition, there is no way an implementation could determine if the type of the parameter was a type not directly convertible to int. The necessity of a diagnostic should not be determined by the presence of a macro, so a diagnostic is not required in the general case.

#108 - (Keaton) Accept suggested response with Plauger's suggested amendment.

#109 - (Guilmette) The issue is undefined behavior at run time vs. compile time.

Guilmette requested the following example be used instead of the one given in the original DR:

```
int array1[5];
int array2[5];
int *p1 = &array1[0];
int *p2 = &array2[0];

int foo()
{
   i = (p1 > p2); /* Must this be "successfully
                     translated"? */
   1/0;           /* Must this be "successfully
                     translated"? */
   return 0;
}
```

MacDonald recalled that at the New Hampshire meeting the committee had decided that the compiler can't refuse to translate because that would require the compiler to know all execution paths to deduce run time undefined behavior.

Guilmette still wants to know if undefined behavior can occur at compile time. The consensus was that yes, but only if the compiler can determine that the given path would be executed.

Agreed to the intent, suggested response wording:

> The presence in a translation unit of an expression which would trigger undefined behavior *if evaluated* may not be used as an excuse for an implementation to fail to "successfully translate" the give translation unit unless the expression in question appears in a context where a constant expression is required. (See subclause 6.4; Semantics, and the associated footnote.)
>
> For an expression which is defined to yield undefined behavior (e.g. 1/0), if the expression appears in some context where a constant expression is *not* required, the undefined behavior acutally arises *only if* the expression in question would be evaluated (at run-time) according to the abstract machine semantics. (See subclause 5.1.2.3.)

#110 - (Jones) Replace suggested response with:

> No diagnostics are required for any of the above declarations. Each of the function definitions or declarations render the translation unit not strictly conforming. See also DR#047.

#112 - (Zeeb) Agree with the suggested response, but append "Subclause 6.3.8 makes it clear that the behavior here is undefined."

#113 - (Keaton) Accept the suggested response.

Jones noted, however, that 6.6.4 is misquoted. Will be fixed.

#115 - (Jones) Modify suggested response as follows:

Remove explicit answers.

Insert after 1st sentence: "Hence a diagnostic is not required, but a program that uses such a form has undefined behavior."

Remove last sentence.

#117 - (Zeeb) After resolving DR#87, agreed to suggested response:

The standard does not forbid an implementation from interleaving the subexpressions in the given examples as specified above. Similarly, there is no requirement that an implemenation use this particular interleaving.

The fact that one particular interleaving yields code that properly delimits multiple modificatons of the same object with sequence points is irrelevant. Any program that depends on this particular interleaving is depending on unspecified behavior, and is therefore not strictly conforming.

#118 - (Keaton) Suggested response:

No, diagnostics are not required. Corrected in response to DR#13, question 5.

#119 - (MacDonald) Suggested response:

(A) No.
(B) No, this is undefined behavior because "array" is not an array since its element type is not an object type, therefore, 6.5.7 does not apply.

#120 - (Jones) Suggested response (edited from Feather):

Subclause 6.5.2.1 states "A bit-field is interpreted as an integral type consisting of the specified number of bits". Thus the type of `object1.bit` and `object2.bit` can be described as "`unsigned int:1`". When a larger integer is converted to this type, it is done according to the rules in 6.2.1.2. Thus the value 3 is converted to the value 1.

#121 - (Gwyn) Conversion of pointer values to integral types.

Jones pointed out that by DR#57, no type may be long enough. Plum and Mooney argued why make useless behavior more precise–always implementation defined. Guilmette believes it is still better to be more precise. Jones suggested using part of Feather's response.

Suggested response, edited from Feather's reply:

The "size required" is that required by the implementation. The words "If the space provided is not long enough" make it clear that it is the size of

the type that is relevant, and means that any type which is at least as long as the type of the "size required" is also acceptable.

The size required need not be related to the results of "sizeof" applied to the expression.

#122 - (Zeeb) Suggested response:

See DR#15. " The original type" applies to both width and signedness. `object.bit` promotes to int, and the program prints 1.

Plum notes that we should revisit type with respect to bit fields during our standards revision process, though.

#123 - (Benito) Basically agreed with Feather's suggested response, edited as follows:

A: Yes
B: Yes

As stated in 6.5.3, "The properties associated with qualified types are meaningful only for expressions that are lvalues."

The definition of "type category" is given in subclause 6.1.2.5, p.24.

#124 - (Zeeb) Suggested correction to 6.3.4–insert: "a":

Unless the type name specifies a void type, ...

#125 - (Jones) Use suggested response from Feather:

Applying `&` to an identifer of type "const void" is undefined behaviour, as explained in the response to DR012Q1. Thus an implementation can define any semantics it wishes. A strictly conforming program cannot contain such a construct.

#126 - (Gwyn) Agree with Feather's email response, with some edits:

A typedef introduces a name for a type. This is not a macro, and the type must indeed by "magically parenthesised". In:

```
typedef int * ip;
ip x;
const ip y;
```

the type of `x` is "pointer to int", and the type of y is "const pointer to int". This is exactly analogous to the fact that:

```
ip x1, x2;
```

declares both `x1` and `x2` as having the type "pointer to int" and is not to be read as "`int *x1, x2`".

Guilmette asked if we needed to clarify wording in the standard. Most felt that a careful reading of the standard was sufficient. Plum pointed out that it was an unending task to clarify clarifications.

#127 - (Zeeb) Suggested response:

> See DR#13, question 3. There is no requirement that the composite type be unique, and either type can be the composite type.

#128 - (Plum) Suggested response:

> Yes, line 7 violates the semantic rule cited.
>
> Yes, `struct TAG` represents an incomplete type.
>
> The application of rules such as scope rules need not be re-stated at each relevant point in the standard.

Guilmette objected to not adding clarification of "later" to "later in same scope".

Note: correct reference from 6.5.3.2 to 6.5.2.3 in the DR.

#129 - (MacDonald) Suggested response:

> Q1. No.
>
> Q2. No change necessary because 2nd bullet of 6.1.2.3 states name spaces of tags are shared, and therefore inner "enum TAG" hides outer "struct TAG", and therefore the case "(struct TAG *)" attempts to declare a new "struct TAG" and thus constraint in 6.5 is violated.

#130 - (Jones) Accept suggested response.

#131 - (Gwyn) Standard does not have words that state that const-qualified array member of a struct forces the struct to be const-qualified, so need to add.

MacDonald stated that 6.5.3 as is makes cited behavior undefined, even though a diagnostic would not be issued (no constraint violation). Gwyn stated that he wants a diagnostic.

Plum clarified that const percolates down to array element type. J16 added explictly that an "array of const" is a const type. The C standard doesn't say that, but if it did, wouldn't need the suggested change. MacDonald asked if you take the address of a const member (`&s.a`) you get a pointer to a const; can you assign that to a pointer to a non-const? No, incompatible types.

Guilmette noted a second problem, which could be fixed at the same time: going from struct to element type, e.g.,

```
struct { int a[5];} const s1;
```

After more discussion, Gwyn's suggested correction was accepted with edit to apply to "all contained aggregates *or unions*."

#132 (Mooney) Accept the suggested response, but add:

> The translation unit must be successfully translated.

#133 - (Zeeb) Suggested response:

> The C standard is sufficiently clear that the described behaviors are undefined.

Note: Feather reported that he is not the source of this DR, and does not wish to be associated with it.

#134 - (Mooney) Suggested response:

> From subclause 7.1.4, an error number is a positive int value that may be assigned to errno by library functions. Since **strerror** maps an error number into an error message string, it need only do so for the possible values of an error number. The behavior is undefined if the argument is not an error number.

Jones suggested that the issue was undefined vs. unspecified behavior; if not a valid error number, it is a domain error. So add to the above:

> If errnum is not an error number, it's a domain error for **strerror**, and the behavior is undefined. For "zero", behavior is undefined; for "EDOM" and "ERANGE", behavior is defined; for "any value that a library routine might set errno to", behavior is defined.

#135 (Jones) Use edited suggested response from Feather:

> There are no zero length objects in C. Therefore, if the size argument to **fwrite** is 0, this is outside the domain of the function and (by 7.1.7) the result is undefined. The C standard is not in conflict with the cited behavior of SVR4.

#136 (Farance) What are **mktime** options when tm_isdst member is -1 (unknown). In the spring, there is a hole; in the fall, overlap. There are 3 choices:

> (1) return -1; no one wants to return -1 becaue value is representable.
> (2) return either standard or daylight time - ambiguous time - as long as implementation gets either one right.
> (3) return "normalized" result, similar to handling of "holes" in month days. "holes" mean "take last valid value and add N units."

General normalization rules:

> if x < normal range => subtract N units from first valid value
> if x > normal range => add N units to last valid value
> if x in hole => add N units to last valid value
> if x in overlap => use first value

To application programmers: only reliable way to get "previous local hour" is to:

```
tm_mday--;
tm_hour += 23;
```

Gwyn doesn't agree that the intent of the committee was to tell application programmers to use this prescribed formula. Believes **mktime** is supposed to do the best it can. Also, ambiguity doesn't exist in specification, only in question posed in the DR. Want to avoid a recursive solution. Ambiguity exists if can't determine whether DT or ST in **mktime** when tm_isdst is -1.

Plum believes the standard isn't precise enough to answer the question, and while the suggested formula is interesting for the next standard, we can't object to returning -1 since "the" calendar time must be unique. Gwyn believes that if -1 is returned, should also leave -1 in tm_isdst. Plum agrees that tm_isdst should be coerced to -1 when **mktime** returns -1.

Jaeschke observed that this is then a case of "Yes, the standard does not say."

Suggested response:

> The standard does not specify the behavior precisely enough to preclude **mktime** from return a value of (time_t) -1 and leaving the tm_isdst member set to -1 in such situations.

When tm_isdst is left as -1, this implies that **mktime** could not determine the correct representation rather than no correct representation exists. Could argue this is deliberate ambiguity, or make unambiguous in the next revision.

#137 - (Zeeb) Suggested response:

> As specified in subclause 7.9.6.1 for the + flag, a negative value is being converted, so a minus sign is required. The intent is that the sign is determined prior to conversion.

11. Fortran 90 VLA/Data Parallel Report [N353/94-038]

Farance presented a brief overview of his report on comparison of features from Fortran 90 and NCEG VLA and Data parallel proposals. He identified 3 issues and subissues being addressed by NCEG proposals:

- shape
  - rank and dimensions
  - reshape
  - argument passing
  - function prototypes
  - stand alone arrays: ALO's - values, C arrays - ptr
  - shapeof, shapeis, rankof, alo, carray
- layout
  - distributed memory
  - near, far, huge ptrs:

```
void *x
void layoutis(...) *x;
void volatile layoutis(...) *x;
```

  - layoutof, layoutis

- argument passing
- selectors
  - array slice
  - scatter/gather

A fourth issue is parallelism, to be discussed under separate agenda time. His report compares what happens at function call boundaries (see the table in N353).

Farance identified two types of fatness in fat pointers. (1) Wider than void *. (2) Additional information such as shape => dope vector.

Farance noted that this was just responding to the action item taken at the Kona meeting.

Gwyn noted that in the comparison tables, the Farance proposal appears to provide the most functionality!

12. Future Revision of ANSI/ISO 9899 [N328/94-012, N346/94-031]

Jaeschke led the discussion on his proposed charter for revising the C standard. He noted that he will rule out of order any specific technical proposals for this discussion. What we need to do at this time is establish rules, streamline the process. Also need to identify roles for work to be done.

With this as preface, we reviewed §3 of his proposed charter, question by question.

- User community response

  Jervis noted that the Sun C users are very supportive.

  Gwyn stated that the Unix community has two camps: (1) NIH, give us K&R, and (2) those that accept the standard. Jaeschke asked if supported by vendors. Yes.

  Zeeb stated that he sees the same users as Gwyn plus those that want backward compatibility with VAX C.

  Benito sees lots of interest in NIST standardization.

  So the community response is taken as very positive.

- What to fix? Anything broken? Always broken or have circumstances changed?

  Plum believes all the DR's were pretty tangential, so don't think anything is really broken.

  Gywn added that the TC has fixed those, and are already part of the standard.

  Tydeman reminded us that we are also seeing influence from LIA and other standards bodies now.

Farance believes we need to fix incomplete I/O specification (e.g., file systems).

Jervis made an appeal to keep perspective; we've made minor fixes.

Guilmette cautioned on paranoia about changing the standard in response to DR's; urged us to reword or clarify now. Also, tighten up (remove) as much undefined behavior as possible in this revision to enhance portability.

Gwyn stated that we shouldn't rely strictly on DR's as directives for what needs to be enhanced.

Farance: fix ungetc()! Gwyn: fix errno!

Thomas sees need for support for numerical programming.

Jaeschke will summarize these points in revision.

- Original charter - how has it held up over time?

  Gwyn suggested that we should add to it so that we don't introduce gratuitous differences with C++.

  Plum said we shouldn't change the charter–that would open it to ISO scrutiny.

  Guilmette suggested that the "spirit of C" (2.6) in §2 should have included "keep the language close to the machine". C++ has too much hidden stuff. There was discussion on this point. MacDonald finally suggested that we no longer know what "close to the machine" means; it would be better to say "keep it small and simple."

- C as a general-pupose language; C vs. C++

  Gwyn asked if we really want to encourage C++ use–e.g., add C++ I/O features?

  Jervis wants to avoid C++ extremes like "fascist enums"; tighten up the language and make it safer to use, but keep it efficient.

- What have we learned from DR process?

  Gwyn thinks we should take a philosophical scan over the DR's to get sense of need of community. Also go back over the public comments from the last standard; we rejected many requests that didn't seem worthwhile at the time.

  Plum believes we should use existing implementations as criteria. Gwyn added that they should not be only criteria; should consider exceptions when clearly needed. Plum reiterated that we should still express preference for prior art. Jaeschke noted that we should also consider the type of implementation–e.g., whether academic or commercial.

- Relationship with C++?

Gwyn asked what is the common subset of C and C++. Plum answered that it is as described in Appendix C of the C++ working paper. Gwyn wanted to know how you could guarantee a common subset without a common base document?

Jaeschke suggested there should be a formal committee to coordinate this. Plum believes we should avoid creating committees, and he doesn't think there is sentiment for such a committee in X3.

Jaeschke suggested that, alternatively, the two committees could merge and have two subgroups. Jervis thinks this would be an enormous burden on J16; it's clear that for this revision of C, that such a move would derail their process to produce the C++ standard. Jaeschke agrees it would not be feasible now, but we need a long term goal to drive our revision process.

Gwyn then shared that Stroustrup says it was an accident that he happened to start with C, really wanted a new language.

Thomas wanted to know how big is the document that describes the common subset of C and C++? It is the joint responsibility of the two committees to keep this updated.

Jaeschke asked how do we deal with adding to C features that don't exist in C++? Plum expects C++ will need an amendment immediately to address requests for features from C++ users. Make sure new C features could be something C++ could evolve to. The most they can ask is have we considered liaison implications.

Guilmette noted there are two aspects: technical and political. Gwyn believes that as long as no technical incompatibilities (e.g., different semantics for C++ syntax), political issues are benign.

Jaeschke summarized: (1) we are not constrained by where the C++ standard is now, and (2) we ought to consider impact of our changes.

- Other influences? Looked at §4 of the proposed charter as well
  - add "public review comments" to item 2
  - add XOPEN to item 5
  - item 6 should talk about "Inter-Language" work, not just C++
  - reword item 7 as "Other papers and proposals from the member delegations such as ...".

  Plum said we should ISO-ize the way we phrase our charter document with respect to WG14 and TR's: numerical work of a national body is not proper subject of standard review in SC22's point of view.
- Reviewed rest of Jaeschke's proposed charter.

  Thomas questioned if "running rampant" (§5) was politic.

  In §6 and §7, (1) need standard form or template for submitting proposals, and (2) need submission guidelines and someone to enforce screening and supplying guidelines to submitters.

Walls suggested that we use the word "guidelines" instead of "rules" (§6).

Jervis would like to not introduce formalisms if not necessary. Requiring champions to attend meetings cuts down on a lot of noise.

Jaeschke will coordinate submissions, etc.

Benito volunteered to coordinate the rationale. Expects to produce new document rather than making diffs to old one.

We should provide input on the subject of a redactor to Benito to take to WG14.

* Zeeb, Plum, Gwyn, Keaton, Thomas, and Walls will review Jaeschke's revision of this draft charter.

13. US TAG Meeting [N325/94-009, N359/94-044]

A Benito will serve as head of delegation for the Tokyo meeting.

MSP Move we approve Farance, Walls, Plum, Benito, and Jaeschke as the US delegation to Tokyo. (Plum, Benito)
16/0/0/3/19

We also need a delegation for the Dallas meeting. Plum noted that it is impolite to have a large national delegation (i.e., don't make delegation all attending!). Gwyn asked for clarification: don't have to be in the delegation to vote in US Tag? Yes. The US has just one vote to the ISO committee.

MSP Move we approve the same US delegation to Dallas. (Plum, Benito)
16/0/0/3/19

Plum advised that for certain votes there is a circuitous route because of the ISO/ANSI I-project. (Secretary's note: I'm sure there was more context for this statement, but it didn't get into my notes! Apologies!)

We need to form a position on DAM [N325/94-009]. Plum expressed concern about the cover letter since the review period and the date of the Tag meeting are different–i.e., the tag meeting precedes closing of the review period. He asked if anyone had received any formal comments? No. Informal comments? No.

MSP Move the US supports ISO 9899:1990/DAM1. (Benito, Farance)
14/1/0/4/19

MacDonald asked when this becomes normative. Benito will find out; believes it is the end of the year.

* Benito will post the effective date of the DAM to J11 email reflector when known.

Gwyn asked if any technical changes, must they be part of this vote? Yes.

Jaeschke asked the DPCE committee to report on resolution of request from OMC to take on the C Binding for X3H5. Keaton reported that this had been discussed in the DPCE evening meeting. There had been a prior agreement between X3H5 and DPCE that no overlap existed between the groups. DPCE has no resources to assign to such work. Hence DPCE recommends that we decline to accept this responsibility, unless someone in J11 wants to champion it.

MSP Move that X3J11 decline to take on project 965-D (C Binding for X3H5). (MacDonald, Stanberry)
15/0/1/3/19

* Jaeschke will respond to OMC on declining to take on the X3H5 C Binding.

14. Administration

14.1 Summary of Decisions Reached

Stanberry reviewed the votes taken at the meeting.

There was discussion of the vote taken on VLA's. Was this officially freezing this document for the TR? Consensus was that it was.

Keaton asked for confirmation that the following subgroup proposals are frozen: FP/IEEE, VLA, Designated Initializers, Compound Literals, and Aliasing. Yes.

14.2 Actions Items Committed To

Stanberry reviewed the action items from the meeting, and we added a few new ones.

* Mooney, Gwyn, Farance, and Zeeb will serve as review committee for the <inttypes.h> proposal.

* Jones, Zeeb, Ziebell, Kwan, and Gwyn will email the responses for DR's for their respective subgroups to Gwyn.

* Gwyn will integrate the DR responses and send them to Plauger.

14.3 Future Meeting Schedule

N358 provides the host information for the December meeting in the Dallas area, provided by Convex.

The schedule for WG14 and X3J11 is:

7/27-29/94	Tokyo	Japan Standards	WG14
12/5-9/94	Dallas	Convex	co-located
6/19-23/95	Copenhagen	Danish Standards	co-located
10/16-20/95	Boston	Thinking Machines	co-located
2/5-9/96	Irvine	Unisys	co-located

Mooney will check with IBM on hosting a meeting in Toronto in '96, which will count as a non-US meeting.

DPCE will meet 9/26-28/94 in Boulder, CO, hosted by Keaton Consulting.

VLA will meet 8/22-23/94 in New York, hosted by Farance, Inc.

14.4 Mailings and Submission Procedures

All document numbers must be obtained from Plauger:

P. J. Plauger
398 Main Street
Concord, MA 01742

pjp@plauger.com
phone: 508-369-8489
fax: 508-371-9014

Items for mailings should have the document number on the cover of the document, and then should be sent to Plauger. He prefers hard copy, then fax, then email.

Deadlines for mailings are:

Post-San Jose	7/8/94
Post-Tokyo	8/26/94
Pre-Dallas	10/28/94

Reminder: make sure any papers to be presented formally at the December meeting get into the mailing.

Thomas asked if we could be notified of other proposals to be included in the TR that are expected to be voted on at the next meeting so we can be prepared. Jaeschke stated, by definition, since December is the deadline for the TR, expect to vote on the proposals in the mailing.

Thomas suggested that the document numbers for the proposals be sent to Jaeschke so that they could be reflected on the agenda, which must also be in the mailing.

MacDonald asked if any format requirements exist for the proposals. Jaeschke said they all should use ISO reference numbers, with ANSI reference numbers in parentheses.

14.5 Next Meeting Agenda

Jaeschke will schedule agenda time for remaining NCEG subgroups and regular business as usual.

Plum's term is expiring as vice chairman. He is willing to serve again. Jaeschke will circulate the formal notice for this position.

Jervis asked if there would be agenda time in Dallas for proposed revisions to the standard. That would be up to Plauger, and is probably premature at the next

meeting. Plum stated that we have permission to begin discussing revisions so we should plan on it.

14.6 Other Business

There was further discussion of remaining DR's.

Comments on DR#106 should be sent to Jones.

Noted that DR#84, DR#103, and DR#104 are still open.

MSP Move that we accept the committee's responses to the DR's as the X3J11 position. (Benito, Farance)
12/0/0/7/19

14.7 Adjournment

MS Move to adjourn. (Gwyn, Jones)

Without objection, the chair so orders.

The meeting was adjourned at 12:18 PM, 10 June.