C 44

ISO/TC97/SC22
Languages
Secretariat: CANADA (SCC)

ISO/TC97/SC22

N298

February 1987

TITLE: Summary of Voting and Comments Received on a proposal to register document 97/22 N260-First Working Draft on Programming Language C (Draft Proposed American National Standard) as a Draft Proposal

SOURCE: Secretariat ISO/TC97/SC22

WORK ITEM: 97.22.20

STATUS: New

CROSS REFERENCE: 97/22 N260

DOCUMENT TYPE: Summary of Voting

ACTION: See attached

Note to Panel Members: 18.3.87

Note the Secretariat action overside. WG-14 is requested to review and respond to the Japanese comments. Obviously any U.K. C Panel work to resolve these issues will be useful input to the June meeting of WG14, so I suggest we discuss these as well as U.K. comments electronically and at our May meeting.
Cornelia

Address reply to: ISO/TC97/SC22 Secretariat
J.L. Côté, 140 O'Connor St., 10th Floor
Ottawa, Ont., Canada K1A 0R5
Telephone: (613)957-2496 Telex: 053-3336

Summary of Voting and Comments Received on a proposal to register document 97/22 N260- First WD on Programming Language C as a Draft Proposal

A Letter Ballot (attachment to 97/22 N260) was circulated to SC22 with a return date of <u>1987-02-13</u>.

The following responses have been received:

'P' Members approved the registration of 97/22 N260 as a DP:	12(Austria, Belgium, Canada, Czechoslovakia, Finland, France, Italy, Netherlands, Sweden, UK, USA, USSR)
P' Members approving with comments:	1(Netherlands)
P' Members disapproving the egistration of 97/22 N260 as a DP:	1(Japan)
P' Members having abstained:	0
P' Members not voting:	3(China, Germany FR, Norway)
O' Members approving the egistration of 97/22 N260 as a DP:	1(Romania)

<u>omments:</u>

ttachment 1 - Japan
ttachment 2 - Netherlands

<u>ecretariat action:</u>

ince the majority of SC22 'P' Member Bodies have approved the egistration of 97/22 N260 as a DP but noting that one disapproval as been registered, the SC22 Secretariat will:

- Request SC22/WG14-C to review the Japanese comments in view of resolving the negative vote.

- Request SC22/WG14 to prepare a response to the Japanese comments.

- Based on SC22/WG14's recommendation, the SC22 Secretariat will forward either 97/22 N260 or a revised version prepared in consideration of the Japanese comments, to ISO/CS for registration as a Draft Proposal.

Japanese Comments on SC 22 N 260

We think that the First Working Draft (97/22 N260) is far clearer.more consistent and unambiguous than the basic documents. We have a high regard for the effort of the ANSI commitee ,and approve the First Working Draft in principle.

However we found out a lot of unclear.inconsistent and ambiguous points in the First Draft during our review. so we cannot approve the registration of document 97/22 N260 as a Draft Proposal. We think it is necessary to continue brushing up document 97/22 N260.

We attach Japanese comments on the First Working Draft. We expect the response for each our comment.

P.9 TITLE: Syntax notation
PAGE : 17 LINE : 3

EXPLANATION:
A colon(:) is also used as a meta constant in the proposed standard. Therefore, the colon defined here as a meta symbol is ambiguous.

PROPOSAL:
The sentence in the line 3(ie. A colon following ~ .) should be :

A colon(**:**) following ~ .

P.10 TITLE: Tokens and preprocessing tokens
PAGE : 17 LINE : 27

EXPLANATION:
The discussions of tokens and those of preprocessing tokens are intermixed and are difficult to understand. For example, the following question cannot be easily solved.

```
#define   a   -x
-a
```

Is the result --x (decrementing x) or - -x (two unary minus operators)? According to the Rationale, the committee seems to employ the second interpretation. Namely, preprocessing tokens resulting from the preprocessing phase are directly interpreted as normal tokens without splits or joins. We agree with this token-lvel macro processing instead of the conventional character-level macro processing.

However, we cannot derive this conclusion from the text of the proposed standard. Since the discussions of token separators are given in 3.1 independently of the preprocessing, we easily misunderstand that the tokens are parsed from scratch after the preprocessing phase. 2.1.1.2 gives no help about this point.

PROPOSAL:
Discussion about token separators should not be given in 3.1. These discussions should be given in 3.8. 3.1 should only mention that the resulting preprocessing tokens are directly used as normal tokens.
We think that 3.8 should not be placed at the last section of the language definition. Since it is the first phase of the compilation, it should be discussed before the section of lexical elements.

Chapter 1 Proposals

P.1 TITLE: Conforming Implementation
PAGE : 3 LINE : 7

EXPLANATION:
The term "conforming implementation" is not defined.

PROPOSAL:
The following definition should be given.

- A conforming implementation is a conforming hosted implementation or a conforming freestanding implementation.

P.2 TITLE: Definition of preprocessing token
PAGE : 5 LINE : 37 ~ 38

EXPLANATION:
In the translation phase 3, the proposed standard says the source file is decomposed into preprocessing tokens and sequences of white-space characters. The "preprocessing token" used here is considered to be distinct from the syntax element "preprocessing-token" defined in §3.8. However, the term "preprocessing token" is not explicitly defined.

PROPOSAL:
Should be explicitly defined for the preprocessing token in §3.8.

P.3 TITLE: Recursive process of the trnalation phase for a #include preprocessing directive
PAGE : 5 LINE : 43 ~ 45

EXPLANATION:
It's not clear that to which phase a #include preprocessing directive causes the named header or file to be processed from phase1, recursively.

PROPOSAL:
We propose to describe that "a #include preprocessing directive causes the named header or file to be processed from phase1 to phase4. recursively".

P.4 TITLE: Arguments to main function in a hosted environment
PAGE : 7 LINE : 3 ~ 5

EXPLANATION:
The proposed standard describes that "If the hosted environment cannot supply strings with letters in both upper-case and lower-case, the implementation shall ensure that the strings are received in lower-case."
In this description, it is impossible that strings are received in upper-case.

PROPOSAL:
Considering that the strings in upper-case are received to main function, we propose the following description.
"The implementation shall supply a means to pass the strings in lower-case."

P.5 TITLE: Definition for conforming environment
PAGE : 7 LINE : 48

EXPLANATION:
The definition for the term "conforming environment" is not given.

PROPOSAL:
The term is initially used here. Therefore, the term should be written in italic characters and it's definition should be given.

P.6 TITLE: Signal handler
PAGE : 12 LINE : 5

EXPLANATION:
The definition for the term "signal handler" is not given. There is a forward reference § 4.7.1.1. However, no definition for the term is given there either.

PROPOSAL:
The definition for the term should be given.

P.7 TITLE: Translation limits of pointer, array and function declarators
PAGE : 12 LINE : 26

EXPLANATION:
It is ambiguous whether the translation limits, 12, means the total number of pointer, array and function declarators, or the number of each declarator.

PROPOSAL:
The description on page 12 should be changed to the following one same as line 43, 44 on page 56; "12 pointer, array, and function declarators (in any combinations) modifying a basic type in a declaration".

P.8 TITLE: Parameter and argument
PAGE : 12 LINE : 33 ~ 34

EXPLANATION:
The term "parameter" and "argument" is not used distinctively.

PROPOSAL:
An alternative description of line 33~34 is:

- 31 parameters in one function definition and 31 arguments in one call
- 31 parameters in one macro definition and 31 arguments in one invocation

P.11 TITLE: Scope of tag and enumeration constant

EXPLANATION:

(1) page 19 , line 14 ~ 17

AS "tag" and "enumeration constant" are not derived from "declarator", the following description does not apply to the definition of their scopes.

"All other identifiers have scope determined by the placement of their declarations." If the declaration appears outside any block, the identifier has file scope, which extends from the completion of its declarator to the end of the source file. If the declaration appears inside a block or in the list of parameter identifiers in a function definition, the identifiers has block scope, which extends from the completion of it declarator to the } that closes the associated block."

(2) page 53 , line 5   page 54 , line 34

The term "declaration" is defined as a syntax element, but it seems that the "declaration"s in the following descriptions are not used with the strict meaning of the syntax element. As the result, the beginning of their scopes are unclear.

"A subsequent declaration in the same scope may then use the tag, but the bracketed declaration list shall be omitted."

"The scope of an enumeration constant begins after its declaration and ends which the scope of the enumeration of which it is a member."

PROPOSAL:

The scopes of "tag" and "enumeration constant" should be defined explicitly.

P.12 TITLE: Escape sequences of octal integer and hexedecimal integer
PAGE : 24 ~ 25

EXPLANATION:

(1) It cannot be determined by the syntax of the escape sequence \o, \oo and \ooo whether the sequence \123 must be taken as "\1,2,3", "\12,3" or "\123".

(2) Non terminals o and h are not defined.

PROPOSAL:

(1) Add the following descriptions;

The octal integer is the longest sequence of the less than or equal three octal digits that follow the backslash.

The hexadecimal integer is the longest sequence of the less than or equal three hexadecimal digits that follow the backslash and the letter x.

(2) Non terminal o(or h) in the escape sequence \o, \oo and \ooo (or \xh, \xhh and \xhhh) should be defined.

\

P.13 TITLE: Backslash in 3.1.3.4 Character Constants
PAGE : 25 LINE : 7 ~ 9

EXPLANATION:
The "Description" describes that the single-quote (') shall b
ed by the escape sequence \', but it does not describe about
backslash \. The backslash shall also be represented by the e
sequence.

PROPOSAL:
Insert the following description.
"the backslash \ shall be represented by the escape sequenc

P.14 TITLE: Definition of "length"
PAGE : 28 LINE : 34

EXPLANATION:
There is no definition of the term "length".

PROPOSAL:
Should define "length" in terms of "bit" or "byte".

P.15 TITLE: Usual arithmetic conversions
PAGE : 29 LINE : 27 ~ 28

EXPLANATION:
It is not clear what the following description says.
"the type of the result is not changed therby".
The above description is interpreted as a part of the phrase
"~, provided neither~". If the above interpretation is corre
description seems to be redundant for a standard.

PROPOSAL:
The following paragraph should be deleted from the propose
"Operands may be converted to other types, provided neithe
precision is lost thereby; the type of the result is not ch
thereby."

P.16 TITLE: Explicit conversions
PAGE : 29 LINE : 45

EXPLANATION:
There is no definition of "explicit conversions". Does "ex
conversions" mean "cast" operation ?

PROPOSAL:
The definition of "explicit conversions" should be given.

P.17 TITLE: Definition of "regrouping"
PAGE : 31 LINE : 11 ~ 16

EXPLANATION:
There is no definition of the term "regrouping".

PROPOSAL:
Should explicitly define for "regrouping".

P.18 TITLE: Automatic conversions of function designators
PAGE : 31 LINE : 33

EXPLANATION:
In 3.3.0.1, the proposed standard says that a function designator which is an expression is converted to a pointer type automatically. In 3.2.2.1. it says that only identifiers having function types are converted. We think that these two rules are conflicting.
Moreover, we cannot understand what the phrase "where a function designator is permitted" in 3.2.2.1 means. A function designator can appear even when the context requires a pointer to function type. The phrase obviously does not consider this case.

PROPOSAL:
Should give a consistent description.

P.19 TITLE: Types "affected" by default argument promotions
PAGE : 34 LINE : 18

EXPLANATION:
The word "affected" is not precisely defined.

PROPOSAL:
Avoid the word "affected". For example, the rule may be given by counting up the types "affected" by the default argument promotions.

P.20 TITLE: Definition of "common initial sequence"
PAGE : 35 LINE : 12

EXPLANATION:
The definition of the phrase "common initial sequence" is not given. There are two possible interpretations. First, we can consider that if types of initial members are the same two struct's share a common initial sequence. Alternatively, we can think that the matching of member names is required in addition to the first interpretation. In this interpretation, "sequence" is interpreted as "token sequence". Which is true?

PROPOSAL:
Should give a explicit description.

P.21 TITLE: Subtraction of a pointer from an integer
PAGE : 40 LINE : 41

EXPLANATION:
We think that the subtraction operation of a pointer value from an integer value should be inhibited. Currently, the Constraints of section 3.3.6 does not give this rule.

PROPOSAL:
Should give a rule to inhibit "Integer — Pointer" operation.

P.22 TITLE: Subtraction of a pointer value pointing outside of an array
PAGE : 41 LINE : 18

EXPLANATION:
The proposed standard says that a pointer value pointing just past the end of an array can be an operand of a subtraction operator only when the other operand is a pointer value pointing the last element of the array. This restriction is too severe. It does no harm to subtract pointer values pointing somewhere in the array from the just-out-of -array pointer value. For example, if the following declarations are given,

```
    int  a[10];
    int *p = &a[10], *q = &a[5];
```

The subtraction

```
    p - q
```

should be allowed.

PROPOSAAL:
Relax the rule so as to allow the above kind of subtractions.

P.23 TITLE: Nonzero value of pointer types

PAGE :	44	LINE :	29
PAGE :	44	LINE :	46
PAGE :	45	LINE :	16
PAGE :	66	LINE :	34
PAGE :	67	LINE :	23

EXPLANATION:
The word "nonzero" is inappropriate when the expression has a pointer type. There are no "zero pointer value"s. Instead, there are null pointer constants.

PROPOSAL:
Replace the sentence by the following: ... if its value is nonzero when the expression has an arithmetic type, or if its value is not a null pointer constant when the expression has a pointer type, ...
Alternatively, define the words "zero" and "nonzero" for pointer types.

P.24 TITLE: Definition of register storage class specifier
PAGE : 50 LINE : 12 ~ 15

EXPLANATION:
We propose that the definition of register storage class specifier should be changed to as follows. Because lines 12 to 15 in page 50 do not go well with the standard, because this part does not effect directly to programmer and implementer. Only meaningful description is " & cannot be applied". Lines 12 to 15 are concluded from this description.

PROPOSAL:
A declaration with storage-class specifier <u>register</u> is an auto declaration, and the unary operator <u>&</u>(address of) operator shall not be applied to an object declared with storage-class specifier <u>register</u>.
"with suggestion that ... implementation-defined" should be placed in footnote.

P.25 TITLE: Type scpecifiers
PAGE : 50 ~ 51 LINE : 45(page 50) ~ 7(page 51)

EXPLANATION:

We cannot understand which type specifier may be in conjunction with other type specifiers each other, and which lexical order is permitted.
And it is not specified that const can connect with volatile (in spite of the example in page 55 which describes that const can be used in conjunction with volatile).

PROPOSAL:

Should give a table which lists up all possible combination except for const and volatile.(See eg.1)
And should specify that const may be in conjunction with volatile.

eg.1
Possible combinations of type specifiers

```
short int
long int
unsigned int
unsigned short
unsigned long
signed int
signed short
signed long
unsigned short int
unsigned long int
unsigned char
signed short int
signed long int
signed char
long double
```

P.26 TITLE: Bit field in union
PAGE: 52 LINE: 19 to 23

EXPLANATION:
We think that there is no reason to permit the bit-field in union. The bit-field in union should be forbidden as formar version of the Working Draft.

PROPOSAL:
The bit-field shall not be specified in union.

COMMENT:
The description in lines 19 to 23 cannot apply to the bit-field in union. "to the union in which it reside" in line 37 is misleading description, because any bit-fields cannot be refered by any pointer.

p.27 TITLE: Explanation of types
PAGE: 56 LINE: 29 and after

EXPLANATION:
The explanation for the type in this Working Draft is difficult to understand and is not precise. This Working Draft intend to define the type only to identifier. Consider following example:

```
int *f();
```

In this Working Draft, the type of f is defined as follows
f has type "function returning T1",
T1 has type "pointer to T2",
T2 has type int,
so f has type "function returning pointer to int". But in this Working Draft the connection between D and D1 is not clear.
And we think it is more useful to define inter-mediate declarator such as f(), *f(), because when same form is appeared in expression, they has the same type as declarator. So we propose following style of definition.

PROPOSAL: (underlined words mean italic)

3.5 DECLARATIONS
add following statement to semantics.

Each *declarator* in declaration has the type specified by *type-specifier* in the *declaration*.

3.5.3 Declarators
Change syntax of declarator as follows (suffix is used only for identification).

declarator$_1$:
direct-declarator
pointer *declarator*$_2$

And add following description to semantics.

If declarator has type T and sysntax of declarator$_1$ is direct-declarator then direct-declarator has type T. If declarator$_1$ has type T and syntax of declarator is "pointer declarator$_2$" then declarator$_2$ has type "type-specifier pointer to T".

If direct-declarator$_1$ has type T, and

if direct-declarator is parsed as identifier, then the identifier has type T,

if direct-declarator$_1$ is parsed as "(declarator$_2$)", then declarator$_2$ has type T,

if direct-declaraor is parsed as "direct-declarator$_2$[constant-expression$_{opt}$]", then direct-declarator has type " array of T",

if direct-declarator$_1$ is parsed as "direct-declarator$_2$(parameter-type-list)" or "direct-declarator$_2$(identifier-list$_{opt}$), then direct--declarator$_2$ has type "function returning T".

COMMENT:

"type-specifier T" at line 39 in page 56 may be error, because type-specifier means such as "int" and T is also such as "int". And that "type specifier" may be also used erroreously at line 10 and 42 in page 57, and at line26 in page 58.

P.28 TITLE: Definition of the sequence rules of the abstruct machine
PAGE: 55 LINE: 22 to 24

EXPLANATION:

The proposed standard says that any expression referring to a volatile object should be evaluated strictly according to the sequence rules of the abstruct machine. However, the definition of the sequence rules is not given.

PROPOSAL:

Should be explicitly defined for the sequence rules in section 2.1.2.3.

P.29 TITLE: Multi-dimensinal array in "3.5.3.2 Array declarators"
PAGE: 57 LINE: 34 to 44

EXPLANATION:

The following description is redundant for a standard, because this definition will be derived from the former syntax.

"when several "array of" specifications are adjacent, a multi-dimensional array is declared."

PROPOSAL:

Move this description to a footnote.

P.30 TITLE: declarator in type definition
PAGE: 56 LINE: 29

EXPLANATION:
How the identifier KLICKSP() in the examples in line 47, page 60 can be derived from syntax definitions in the section 3.5 is confusing.
Though there is a forward reference to 3.5.5, the confusion occurs from the fact that semantics of identifiers in 3.5.3 give a mixed-up description of essentially different two things, namely for type definitions and for declarations of object or function.

PROPOSAL:
The semantic descriptions for each of the following two cases of an identifier appeared in declarators should be given separtely.
(1). An identifier declared as a new type name in a type definition
(2). An identifier declared as an object or function

P.31 TITLE: Character array object whose members are initialized with string literal
PAGE: 63 LINE: 1 to 6

EXPLANATION:
It is ambiguous specification that the declaration
```
    t[3] = "abc";
```
is identical to
```
    t[] = {'a','b','c' } ; .
```

PROPOSAL:
when character array size is equal to string literals, common warning message may be generated to indicate lack of null character.

P.32 TITLE: Terms "compound statement" and "block"
PAGE: 65 LINE: 13

EXPLANATION:
Two different terms, "compound statement" and "block", are used to express the same concept. That is, term "block" is used in section 3.1.2.4 and "compound statement" is used in section 3.6.2.

PROPOSAL:
One of the two terms should be used to express the <u>block</u> concept.

P.33 TITLE: definition of "h-char"
PAGE: 74 LINE: 14 to 16

EXPLANATION:
If the hierarchic file system must be described with '>', the header-name cannot be specified.

PROPOSAL:
The definition of "h-char-sequence" should be implementation-defined.

P.34 TITLE: Fflush function
PAGE: 113 LINE: 15

EXPLANATION:
A function returning value in case a write error does not occur is not specified.

PROPOSAL:
Should be added a clause ",otherwise zero".
(The same holds for fseek, feof and ferror functions.)

P.35 TITLE: The amount of padding specified by the precision
PAGE: 116 LINE: 9 to 10

EXPLANATION:
The description, "The amount of padding specified by the precision overrides that specified by the field width", is ambiguous.
If the result of the execution of the statements, "int i=1; printf("%6.2d\n",i);", is "   01", the above description is insufficint to define the standard.

PROPOSAL:
Should define "amount of padding specified by the precision" more precisely.

P.36 TITLE: Signed decimal notation,etc.
PAGE: 116 LINE: 39 to 40

EXPLANATION:
Signed decimal, unsigned octal, unsigned decimal and unsigned hexadecimal notations are used for defining output format. But thier definitions are not given in this Working Draft.

PROPOSAL:
Should define above notations explicitly.

P.37 TITLE: Vfprintf function
PAGE: 123 LINE: 30 to 34

EXPLANATION:
The side effect on the third formal parameter 'va-list arg' seems to be produced by vfprintf function, but it is not well specified.

PROPOSAL:
Should specify the effect on the 'arg'.
(The same holds for vsprintf and vprintf functions.)

P.38 TITLE: Definition of "next character"
PAGE: 125 LINE: 10

EXPLANATION:
A term "next character" is not defined. Usually "next character" will be indicated by the file position indicator, but it is not always true.

PROPOSAL:
Should give the definition of "next character" on the appropriate place.

P.39 TITLE: Fputc and file position indicator
PAGE: 125 LINE: 45 to 48

EXPLANATION:
The fputc function puts a character "at the position indicated by the associated file position indicator (if defined)".
The explanation of append mode "a" for fopen function specifies that "opening a file append mode causes all subsequent write to the file to be forced to the current end-of-file, regardless of previous calls to the fseek function".
Consider the following case.

```
file = fopen(filename,"a+");
/*  ... */
fseek(file,x,y);
c = fgetc(file);
/*  ... */
fseek(file,x,y);     /* File position indicator is well defined*/
c1 = fputc(c,file);
```

In the above example, "fputc" puts the character at the end-of-file or somewhere else?

PROPOSAL:
Should add such as following statement in the "Description".
If the stream is opened with an append mode, the character is appended to the output stream.

P.40 TITLE: Clearerr function
PAGE: 131 LINE: 26 to 27

EXPLANATION:
The fseek function also clears an end-of-file indicator.

PROPOSAL:
Should replace the word "these indicators" by "both of these indicators", and add following statement in "description" or as a footnote.

The end-of-file indicator is also cleared by the fseek function.

P.41 TITLE: Memory management function
PAGE: 138 LINE: 2 to 6

EXPLANATION:
The effect of referring to the space allocated by malloc and realloc functions before the value is assigned is not specifically explained.

PROPOSAL:
Should add the following statement .
If the value of object on the space allocated by malloc or realloc function is used before one is assigned. the behavior is undefined.

P.42 TITLE: System function
PAGE: 141 LINE: 7 to 8

EXPLANATION:
The return value in case of null string and of nonexistence of a command processor is not specified.

PROPOSAL:
Should add the clause ",or zero to indicate that there is not a command processor".

Chapter 2 Qestions

Q.1 TITLE: Translation limits
PAGE: 12 LINE: 21 to 22

QUESTION:
What is the meaning of the following description?
"The implementation shall be able to translate and excute at least one program that contains at least one instance of every one of the following limits."
Does this mean that
"The implementation shall be able to translate and excute a program which contains instances of every limits."
or
"the implementation need not be able to translate and execute a program which contains more than one instance of the limits."
?

Q.2 TITLE: Are infinite indirections of a function allowed ?
PAGE: 31 LINE: 33

QUESTION:
An expression having a function type may be automatically converted to the corresponding pointer type. According to this rule, when a function "f" is defined, the following function calls are all valid.

```
f()
(*f)()
(**f)()
(***f)()
(****f)()
 ...
```

In the second call,the identifier "f" has a function type. It is converted to a pointer type, since the indirection operator requies an operand of a pointer type.The result of the indirection has a function type. Since the function name part of a function call must have a pointer type, the result is again converted to a pointer type.For just the same reason, two or more indirection operators may be applied to a function name.Thus, infinitely many indirections may be applied to function name. Is this interpretation is correct?

Q.3 TITLE: What is "same level"?

PAGE: 40	LINE: 18
PAGE: 40	LINE: 47
PAGE: 43	LINE: 24
PAGE: 43	LINE: 40
PAGE: 44	LINE: 16

QUESTION:
According to 3.3, commutative operators may be regrouped arbitararily. This rule may be applied even if there are parentheses between these operators. In this context, we cannot understand what the phrase "same level" means. Though of course it does not stand for the same parentheses level, the reader may misunderstand the rule. What is "same level"?

Q.4 TITLE: Initialization of an array that has automatic strage duration
PAGE: 61 LINE:

QUESTION:
Is it possible to initialize an array that has automatic storage duration ?
Should describe that the initialization of an array that has automatic starge duration is possible or not.

Chapter 3 Editorial

E.1 Index

All meta-variable names including sub meta-variable in a syntax definition refered from the other syntax definitions and descriptions should be added to the index.

E.2 Array subscripting

Should specify the range of "n" as "n ≧ 2" for the semantics of n-dimensinal array.

E.3 Editorial errors

(1) Page 27 Line 5

There is an erroneous forword reference. "3.8.2" should be corrected by "3.8.3".

(2) Page 32 Line 6

"unary expressions(§3.3.3)" should be corrected by "unary operators(§ 3.3.3)".

(3) Page 167

" §3.3.2.13" should be corrected by "§3.3.13".
" §3.3.2.14" should be corrected by "§3.3.14".
" §3.3.2.15" should be corrected by "§3.3.15".
" §3.3.2.17" should be corrected by "§3.3.17".

Nederlands
Normalisatie-instituut

Kalfjeslaan 2
Postbus 5059, 2600 GB Delft
Telefoon (015) 611061
Telex 38144 nni nl

Datum: 87-02-10

Ref.: 97-22/87-06

Comments of the Netherlands accompanying the vote of approval on the registration of document 22N 260 First working draft on Programming language C as a draft proposal

source:NNI

Although the NNI currently votes yes, the incorperation of some new features may cause the change of our vote to no on subsequent issues of the DP.