The Report on the Standard Hardware Representation for ALGOL 68

By Wilfred J. Hansen and Hendrik Boom

Ø. Introduction

At its September, 1973, meeting in Los Angeles, Working Group 2.1 of IFIP created a Standing Subcommittee for ALGOL 68 Support. The January, 1975 meeting of this Subcommittee in Boston discussed at length a standard hardware representation and authorized a Task Force to draft a proposal incorporating the conclusions of that meeting. An initial draft was presented to the June, 1975, meeting of the Informal Information Interchange at Oklahoma State University. Many improvements and alterations suggested at that meeting have been incorporated into this final version. All suggestions were valuable, even those that served only to stimulate discussion. Subsequently, this report was accepted by the August, 1975, meeting of Working Group 2.1 in Munich and forwarded to IFIP.

A standard hardware representation is desirable for several reasons:

- First, together with the Report*, it provides a complete definition of a single language. As implementations have developed their own solutions to the problems of representation, there have arisen many related languages that differ considerably in appearance. To read or write a program for an alien implementation, a programmer has been required to make a considerable mental readjustment of deep habits. One might argue that no precise standards exist for natural language punctuation and typesetting, but the argument does not apply to artificial languages intended to be read by machines.

- Second, processors other than compilers may be defined for ALGOL 68 programs; for example, macro processors, cross-reference programs, and print formatters. Such processors may be used by all implementations only if the tokens they accept are defined by a standard.

- Third, a single representation convention will promote portable programming. This document specifies a minimum character set that every compiler must accept and the maximum that may be used in a portable program. Consequently, program transportation requires only one-to-one transliteration; the transliterator need not determine the extent of strings, comments, and format-texts.

Several goals have been addressed in creating this standard hardware representation: it should require only a small, widely available character set*; it should minimize parsing problems; it (or some subset) should be teachable; it should be possible to write portable programs that process other programs; it should conform to the Report, existing usage, and usage in other languages; and, above all, it should be a practical, congenial means of expressing ALGOL 68 programs. With the exception of three representations {see *3.7} and the "string break" {see *3.1}, an implementation following this document is an "implementation of the reference language" {R9.3.c}.

---

* In this document, "the Report" refers to the Revised Report:
A. van Wijngaarden, et al., Revised Report on the Algorithmic Language ALGOL 68, Acta Informatica, v.5, Fasc. 1-3, Springer-Verlag (Berlin, 1975).
References to it are in the form of "R" followed by a section number. To avoid confusion, references to sections in this report are prefixed with "*".

---

* With the exception of square brackets, the set of worthy characters is a subset of most versions of ISO-code, ASCII, and EBCDIC:

ISO Standard 646: 7 bit coded character sets for information processing interchange. An earlier version of this standard was considered in Lindsey, C. H., "An ISO-code representation for ALGOL 68", ALGOL Bulletin 31 (March, 197Ø), pp. 37-6Ø (corrected in AB 32.1.3).

ANSI, USA Standard Code for Information Interchange (X3.4-1968), American National Standards Institute (New York, 1968).

ANSI, American Standard Hollerith Punched Card Code (X3.26-197Ø), American National Standards Institute (New York, 197Ø) {defines a version of EBCDIC}.

IBM Corp., IBM 14Ø3 Printer Component Description, Order no. GA24-3Ø73, 197Ø {defines the "TN-chain" version of EBCDIC}.

Hansen, Wilfred J., "A Revised ALGOL 68 Hardware Representation for ISO-code and EBCDIC", UIUCDCS-R-73-6Ø7, University of Illinois, Urbana (November, 1973); revised as "An ALGOL 68 Hardware Representation for ISO-code, ASCII, and EBCDIC" (December, 1974).

1. Definitions

Worthy character - one of these sixty characters:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Ø 1 2 3 4 5 6 7 8 9
space " # $ % ´ ( ) * + , - . / : ; < = > @ [ ] _ |

{This document defines a representation of an ALGOL 68 program as a sequence of worthy characters and newlines.}

Base character - a "character" available at an installation. {Each such character is a composite of some set of marks and codes agreed upon by local convention. The input to a compiler is a sequence of base characters.}

{What I see is that, whereas there is only one form of excellence, imperfection exists in innumerable shapes....

The Republic, Plato}

Disjunctor - a typographical display feature {R9.4.d}, the start or end of a program text, or any worthy character other than a letter, digit, or underscore. {Tags and bold words are delimited by disjunctors.}

Adjacent, follow, precede - Two character strings are "adjacent" if there are no intervening characters or typographical display features. If one string is said to "follow" or "precede" another, they are also adjacent.

Bold word -
i) any representation composed of bold-faced letters or digits in the reference language {R9.4} {i.e., bold-TAG-symbols and the representations shown as bold in R9.4.1}, or

ii) a symbol represented by a bold word, or

iii) the characters written for a bold word as specified below {*3.5}.

Tag - a TAG-symbol {R9.4.2.2.a} {"End of file" is a tag.}

Taggle - a nonempty sequence of letters and digits. {As used in *3.5.1, "End of file" has three taggles.}

2. Representation of ALGOL 68 Constructs

For each worthy character an implementation must provide a base character different from the base character for any other worthy character. The mapping between worthy and base characters should be chosen so as to minimize confusion while paying due regard to prevailing usage. {For example, an implementer should avoid assigning a base character to an unrelated worthy character and also avoid using a character to represent something other than that which it represents in the Report.}

An implementation may augment the worthy characters with the twenty-six lower-case letters. The two cases of a letter are equivalent except as provided in *3.1 and *3.5.2. {This equivalence promotes portability; for example, it prevents distinction between tags that differ only by the case of one letter.}

The Report specifies {R9.3.b} that a "construct in a representation language" is obtained by replacing symbols with their representations. In this document, a representation is specified for each symbol in terms of worthy characters. Constructs in the representation language are encoded for communication and computer processing by replacing each worthy character with its corresponding base character and inserting typographical display features {where permitted}.

3. Specific Representations

3.1 String-items

The set of string-items {R8.1.4.1.b} is the set of worthy characters (as possibly augmented with lower-case letters) excluding quote and apostrophe but including the quote-image-symbol and the apostrophe-image-symbol. The intrinsic value of each worthy character is itself; the upper- and lower-case versions of a letter have distinct intrinsic values. The quote-image-symbol is written as two adjacent quotes and its intrinsic value is a quote. The apostrophe-image-symbol is written as two adjacent apostrophes and its intrinsic value is an apostrophe. {A single apostrophe may be used as an escape character in some implementations.}

An additional typographical display feature, the "string break", is provided for use exclusively within string- and character-denotations. It is written as

- a quote, followed by
- one or more typographical display features other than string break, followed by
- another quote.

{When a string-denotation must be continued to more than one line, a string break permits the number of spaces at the end of one line to be indicated and permits the next line to be indented without confusion.}

3.2 Other-Pragmat-Items

Any sequence of characters {worthy or otherwise} may appear as a STYLE-PRAGMENT-item-sequence {R9.2.1.c} except one containing the sequence {including disjunctors} which constitutes the representation of the STYLE-PRAGMENT-symbol itself {because the latter would terminate the pragment}. An implementation may, however, further restrict the sequences of characters allowed in pragmats {but not in comments}.

Four standard pragmat-items are defined: PAGE, POINT, UPPER, and RES {see *3.2.1 for PAGE and *3.5 for the rest}. All implementations must recognize these items at least in the minimal form

STYLE pragmat symbol, item, STYLE pragmat symbol.

Each of these four pragmat-items is written as a sequence of upper-case letters, and may be preceded or followed by typographical display features. {Note that in all stropping regimes a pragmat-symbol may be written as ".PR" followed by a disjunctor.}

3.2.1 Newpage

When the base character representation of a construct is printed by an ALGOL 68 processor, a pragmat containing the pragmat-item PAGE causes the line after the line containing its closing pragmat-symbol to be printed at the top of a new page {possibly after appropriate headers}. {The PAGE pragmat is, however, not a typographical display feature.}

3.3 Typographical Display Features

The typographical display features are space, newline, and string break. {Newline may be a unique base character or a physical phenomenon like end of record. String breaks are allowed only in certain denotations; see *3.1.}

3.4 Style-TALLY Objects

No representations for any style-TALLY-letter-ABC-symbols or style-TALLY-monad-symbols {R9.4.a} are defined by this document.

3.5 Tags and Bold Words

The representation of tags and bold words is determined by the "stropping regime", of which there are three. A new regime is invoked by a pragmat containing one of the pragmat-items POINT, UPPER, or RES, and takes effect following the closing pragmat-symbol. Stropping does not affect the ´STYLE´ of a representation {so in UPPER and RES, ".PR" matches "PR"}. {Some rules below require disjunctors in certain positions. If necessary, these can be obtained by inserting typographical display features.} {In ALGOL 68, tags are distinct only when the concatenations of their taggles are distinct. For example, "end of file" may also be written "endo ffile".}

{"What did the rug, dog, and fish have in common?"

"Each was a car p et."

Works, Mach Tartaruca}

{Examples are shown with each regime. A few, like ".elIF", illustrate usages that cannot be recommended. These usages are allowed because they are orthogonal and they provide a measure of tolerance to unimportant errors.}

3.5.1 POINT Stropping

Bold words.

- A bold word is written as a point (".") followed, in order, by the worthy letters or digits corresponding to the bold-faced letters or digits in the word.

- A bold word must be followed by a disjunctor.

Tags.

- A tag is written as a sequence of {one or more} taggles separated by zero or more typographical display features.

- A taggle is written by writing, in order, the corresponding worthy letters and digits optionally followed by an underscore.

- If a taggle does not end with an underscore, it must be followed by a disjunctor.

{Examples:

```
    Program: .PR POINT .PR .BEGIN .REAL X;
        X := X-1 .END
    Bold:    .BEGIN, .Real, .elIF, .Xl, .abs
    Plain:   BEGIN, Real, end of file,
        end_of_file, Xl, a_b_, a_ b_
    Error:   .BEGIN_, .X_l, .end_of_file,
        a____b, a _ b, a__ b}
```

3.5.2 UPPER Stropping

Tags and bold words are represented as they are in POINT stropping with the addition of these rules:

- Upper- and lower-case letters may not be intermixed in a bold word.

- The point may be omitted from an upper-case bold word if it is preceded by a disjunctor other than a point, by a lower-case letter, or by a digit that is not an "upper-case digit". An "upper-case digit" is one that follows an upper-case letter or an upper-case digit.

- An upper-case bold word need not be followed by a disjunctor if it is followed by a lower-case letter.

- Upper-case letters may be written only in bold words and character-glyphs {R8.1.4.1.c; these are constituents of string- and character-denotations and of pragments}.

{Examples:

```
    Program: .PR UPPER .PR BEGIN REAL x;
        x := x-1 END
    Bold:    BEGIN, .abs, Xl {even in "a3Xl"},
        .a3 {even in ".a3Xl"}, OF {even in
        "reOFz"}
    Plain:   begin, end of file, end_of_file,
        a3 {even in "a3Xl"}, re {even in
        "reOFz"}
    Error:   REAL_, .real_, X_ij,
        return_value_END

    ".aB" is equivalent to ".a B".}
```

3.5.3 RES Stropping

A "reserved word" is one of the bold words specified in R9.4.1 as a representation of some symbol. {See the list in *B. By R9.4.2.2.b, these cannot be redefined and are thus already reserved in another sense.} In the RES regime, tags and bold words are represented as they are in POINT stropping, with the addition of these rules:

- The point may be omitted from a reserved word if it is preceded by a disjunctor other than a point.

- A taggle must be adjacent to an underscore if its letters and digits correspond, in order, to those of a reserved word.

{Examples:

```
    Program: .PR RES .PR BEGIN REAL X; X := X-1
        END
    Bold:    BEGIN, .REAL,  .X1,  Begin,
        .operator, .AMODE
    Plain:   begin_, end_of_file, end_of_file_,
        x1, AMODE, X 1, endo ffile, X_1
    Error:   .BEGIN_, .X_1}
```

3.6 Composite Representations

Where the representation shown in R9.4.1 appears to be composed of two or more consecutive nonletter marks {"", =:, :=, |:, :=:, :/=:}, the representation is the sequence of worthy characters corresponding to those marks.

The representation of any NOTION1-cum-NOTION2-symbol is the representation of the NOTION1-symbol followed by the representation of the NOTION2-symbol. {The NOTION1-cum-NOTION2-symbols are the composite operators mentioned in R9.4.2.2.d,e.}

3.7 Other Representations

Any symbol whose representation in the Report {R9.4} corresponds to some worthy character is represented by that character. {There are no representations for the times-ten-to-the-power-symbol, the plus-i-times-symbol, or the brief-comment-symbol, but the Report provides alternate constructs for all cases where these symbols might be used.}

4. Transput

The transput representations of objects must use only worthy characters {so that input may be prepared and output interpreted without reference to an individual implementation}. The environment enquiries {R10.2.1} depend on worthy characters as follows:

flip:	"T"
flop:	"F"
errorchar:	"*"
blank:	" "

No value is defined for "null character" by this document. Since there are no worthy characters for times-ten-to-the-power-symbol and plus-i-times-symbol, "E" and "I" must be used instead. The two cases of a letter are equivalent when they appear in the transput representation of any value other than one of mode ´character´ or ´row of character´.

As a result of transput and repr, string values may contain characters that do not correspond to worthy characters. This document does not define the actions taken, if any, when such characters are transput. {Ordinarily, most such characters will simply be read and written as single characters, just as will an "A".}

{ Appendices

These appendices discuss the hardware representation, but they are not to be construed as further specification.

Appendix A. Worthy and Base Characters.

A.1 Rationale for worthy characters.

A.1.1 Specific Unworthiness

The following characters were carefully considered as candidates for worthiness, but were rejected for various reasons:

! - because it may be needed as a base character for "|"

\ - because it is not in EBCDIC and "E" is an alternative.

? - no explicit function is assigned in the Report, so it was omitted to limit the size of the worthy set.

¬ ~ - there are severe difficulties with the hardware representations of logical not and tilde: they may be printed as themselves, as each other, or as circumflex, overline, beta, or even up-arrow.

& - with no monad for not or or, ampersand was deleted to reduce the set of worthy characters.

A.1.2 Specific Worthiness

The following were considered worthy, despite disadvantages:

| - because it is crucial to ALGOL 68, despite device problems almost as severe as those for logical not and tilde.

[ ] - they are traditional ALGOL characters (but see *C.2).

% - well-defined meaning and commonly available; moreover, a short snap quiz determined that even some experts cannot remember the bold alternatives for quotient and modulus.

@ - also well-defined and commonly available.

A.1.3 Transput Environment Enquiries

Flip and flop were chosen to be letters rather than digits because the letters have more meaning when these codes represent Boolean values. Neither a string of letters nor a string of digits is easy to read as a representation of a bits value.

The asterisk was chosen as the value of "errorchar" because question mark was unworthy and asterisk is traditional.

A.2 Relationships between Worthy and Base Characters.

An important step in developing this standard was to relate worthy characters to base characters rather than to specific hardware codes. This has several advantages:

- It avoids restricting the standard to any specific character code.
- It makes the implementer responsible for device-dependent decisions, such as the representation of vertical bar (which may be printed on various devices as any one of "|", "!", "¦", ↑, space, ù, or ð).
- By eschewing diphthongs (e.g., "(/" for "[") it facilitates transportation by strict transliteration.
- It specifies a standard external appearance of programs rather than trying to specify a standard internal appearance.

A.2.1 Disallowed Relationships.

If this report specifies one or more representations for some symbol, an implementation should not provide any additional representation for that symbol in the following situations:

a) where there is an existing special character representation for the symbol, or

b) where the new representation would be another bold representation for a symbol that already has a bold representation.

Situation (b) would not increase expressive power, but would increase the potential for confusion. (However, in a variant language {R1.1.5.b}, alternative bold representations might be appropriate.)

Situation (a) would introduce confusion and ambiguity in transliteration of strings. For example, if "%" and "?" both represent the percent-symbol, there is no simple transliteration for "?" in a string.

To avoid similar ambiguity and transliteration problems, implementations should not provide:

- additional style-TALLY-symbols;
- diphthongs specific to the ALGOL 68 environment.

(Thus "(/" should be neither a style-ii-sub-symbol nor a diphthong for "[".)

A.2.2 Permitted relationships.

If system software commonly uses a diphthong for some representation -- such as the diphthong proposed for colon on some systems -- an ALGOL 68 compiler may have no choice but to accept it as a single character. No problem arises as long as the substitution is universal and unambiguous inside and outside strings.

An implementation may specify two or more separate base characters to represent some one worthy character. This may be necessary, for example, if some device lacks "|" and "!" is to be allowed in its stead. The two base characters should be treated as equivalent everywhere except within strings and on program listings, where each should represent itself. When a program is transported it may be necessary to transliterate both base characters to one new character.

Difficulty arises only when trying to export a program that has attempted to utilize the distinction between the two characters. Such a program is not a portable program.

A.3 Super-set Character Sets.

A.3.1 Escape Character.

Some implementations have defined an escape convention for representing extra string-items. This standard does not prescribe any such convention but, if one is used, the apostrophe should be the escape character.

A.3.2 Admissibility of Other Characters.

After adapting the local characters to the worthy characters, an implementer may find he has "unused base characters" that do not map to worthy characters. For each such character C the implementer may choose from the following interpretations:

a) Unused. C is erroneous except possibly inside pragments.

b) As in the Report. If C appears as a representation for some symbol S in the Report and there is no worthy representation for S, then C - if allowed at all - should be a representation for S. Thus, "\", "$_{10}$", ".", "°", "¢", and "&", "¬", "~", "↑", and the other unworthy operators in R9.4.1.c may be used only to represent themselves (unless a desperately small character set forces their use as worthy characters).

c) An unworthy representation. C may represent some symbol for which no nonletter worthy representation is given. For example, "?" could be a skip-symbol.

d) Style-TALLY-monad-symbol. For example, if "?" were not used as an unworthy representation as in (c), it could be a monad. If this option is chosen, C should look like an operator. For example, "{" might make a poor monad.

e) Style-TALLY-letter-ABC-symbol. Care should be taken that C look somewhat like a letter rather than an operator.

f) A typographical display feature. Such an additional feature should usually be ignored in strings (unlike space).

In addition to one of the above, C may be permitted as an other-string-item.

Appendix B. Bold Symbols and Plain Tags.

B.1 Goals of Stropping Rules.

In addition to the goals listed in *Ø, the design of the representations for bold symbols and plain tags was motivated by the following criteria.

a) There should be a small number of stropping regimes to minimize the size of token scanners.

b) For compatibility with North American expectations, at least one regime must be some form of reserved words.

c) Numerous fortunate installations have two cases and desire some form of case stropping.

d) For the sake of tradition, the standard must include at least one regime where all bold words must be stropped.

e) The standard should reduce the possibility of error and enhance the probability of detecting those errors which do occur.

f) Some means of explicit stropping should apply in all stropping regimes so that, among other reasons, pragmat-symbols may be written in a regime-independent manner.

g) Because it is allowed by the Report, there must be some way to represent a tag or taggle that has exactly the same letters as a reserved word.

B.2 List of Reserved Words.

In the RES regime, all bold words listed in R9.4.1 are reserved. There are sixty-one:

at, begin, bits, bool, by, bytes, case, channel, char, co, comment, compl, do, elif, else, empty, end, esac, exit, false, fi, file, flex, for, format, from, go, goto, heap, if, in, int, is, isnt, loc, long, mode, nil, od, of, op, ouse, out, par, pr, pragmat, prio, proc, real, ref, sema, short, skip, string, struct, then, to, true, union, void, while.

Additional bold words may appear in section 9.4.1 of a document defining a superlanguage {R2.2.2.c} or variant {R1.1.5.b} of ALGOL 68. These words should be reserved in an implementation of the modified language. (Programs using them are not very portable anyway.) If a modified language does not give a meaning to some word in the above list, it should nonetheless remain reserved. Only thus can users of a sublanguage be assured of compatibility with implementations of the full language.

B.3 Other Stropping Regimes.

For compatibility with existing installation practice, implementations may implement stropping regimes in addition to those provided by the standard. However, such additional regimes should be invoked by pragmat-items distinct from those in *3.5. All modifications to the defined regimes -- including extensions -- should be avoided because they would inhibit error detection and decrease portability.

B.4 Inside Pragmats and Strings.

To simulate stropping and taggle concatenation, points and underscores may appear in pragments and strings. This may improve the readability of pragments by distinguishing between natural language words and those from ALGOL 68. However, when appearing as string- or comment-items, points and underscores represent themselves and do not indicate stropping.

B.5 Classification of Points.

The following properties of points hold in correct programs. Implementers may find them convenient.

a) Inside a format-text {1Ø.3.4.1.1.a}, but outside any constituent unit or enclosed-clause, a point is a strop if and only if it is followed, first, by one of "co", "pr", "comment", or "pragmat", and next by a disjunctor.

b) A point is not a strop if it is a character-glyph {R8.1.4.1.b}. {Inside a pragmat an implementation may treat a point as a strop.}

c) Elsewhere a point is a strop if it is followed by a letter.

d) A stropped word is always bold.

Appendix C. Portable Programming.

Appendices *A and *B provide considerable latitude for extension of this standard in response to local conditions; however, no implementation will have all these extensions. This appendix discusses the maximum facilities that may be safely employed in a portable program.

C.1 Character Set Descriptions.

The standard is defined in terms of worthy characters in order that program conversion will require only a transliteration of character codes. To facilitate the debugging of such a routine, a program publisher should provide with published programs a file containing the following:

- one or more lines, as necessary, containing all the characters used in the program. This should begin with all of the worthy characters, in the order in which they appear in *1;
- a description of each character.

Each implementer should provide such a file describing the implemented character set.

C.2 Sub- and Bus-symbols.

Nonstandard implementations sometimes restrict the representations for sub- and bus-symbols. For a portable program, two schemes are possible.

a) Use only square brackets. This scheme is preferable because it is the one most likely to be widely portable. Note that every implementation is required to provide base characters for the square brackets, even though the characters provided may not resemble brackets.

b) Use parentheses, but follow this restriction: No local-sample-generator {R5.2.3.1.b} may begin with a style-i-sub-symbol. {This can always be achieved by inserting a local-symbol.} {Any sublanguage with this restriction is easier to parse.}

All implementations of this report will perforce accept programs written according to both of the above schemes.

C.3 UPPER Case.

Some implementations will be unable to support two alphabetic cases. Users with such implementations can usually import programs by converting all the letters to the single case; this succeeds because the standard specifies that both cases of a letter are equivalent in all but two contexts. The first such context is strings; however, as long as the string is intended only for printing, little damage will be caused by converting its letters to a single case. Programmers should be wary of any program whose correct execution depends on the fact that there are two cases of letters in a string.

The second context where case distinction is allowed is in UPPER stropping. A program so stropped is readily converted to POINT stropping, if every bold word is preceded by a blank and followed by a disjunctor. At its simplest the conversion changes "blank, upper-case-letter" to "point, letter", but this may unduly modify the contents of strings. With more complex logic, even programs without blanks before UPPER-stropped bold words can be translated to some other stropping regime, by the recipient. There is, however, the risk that the line length may be increased by the insertion of stropping points or extra disjunctors. It is possible that this may require that some lines be broken if the receiving installation imposes a maximum line length.

C.4 Newlines in Strings.

Some software environments routinely strip trailing blanks from the end of each record; others pad all records to a fixed length; others perform curious mixtures of these procedures. In either case, the number of blanks in a transported string may change if the string includes a newline. To avoid such changes, newlines in strings should appear only in string breaks.

C.5 Other Characters.

A portable program should be written entirely in worthy characters, because only these characters are available in all implementations. With care, however, it is occasionally permissible to use unworthy characters. For example, unworthy characters can be used in messages intended solely for output. Transliteration of such a character may hinder interpretation of the output, but it will not otherwise affect execution of the program. In particular, "?" and "&" are available in most character sets, so they will cause little difficulty if used within strings.

In any case, if unworthy characters are used, sufficient explanation must be provided to enable correct adaptation of the program to a new character set.

C.6 Character Code Dependence.

Use of <u>repr</u> should be severely restricted. Programs should not depend on the particular character code used by the implementation. This can be accomplished with cautious use of the environment enquiry <u>abs</u>. For example, an array, "char type", to be used to distinguish between letters, digits, and all other characters, could be defined and initialized as follows:

```
[0 : max abs char] int char type;
int kletter = 1, kdigit = 2, kother = 0;
for i from 0 to max abs char
do
   char type[i] := kother
od;
for i to 10
do
   char type[abs "0123456789"[i]] := kdigit
od;
for i to 52
do
   char type[abs
         "abcdefghijklmnopqrstuvwxyz"
         "ABCDEFGHIJKLMNOPQRSTUVWXYZ"[i]]
      := kletter
od
```

{This succeeds even if the receiving installation lacks lower case, because the lower-case letters will have been translated to upper case.}

C.7 Portability of Compiler Character Codes.

Four worthy characters -- "|", "_", "[", and "]" -- are often coded differently, even at installations which nominally use the same character code. Implementors should consider whether to provide means enabling each installation to choose codes for these characters for use in error messages, machine-readable documentation, programs, and normal transput.

C.8 Reserved Words.

Although not allowed by this report, some implementations may have reserved word lists that differ from the list in *B. A portable program using RES stropping should ignore the local list by explicitly stropping words not on the official list and placing underscores adjacent to plain taggles that appear on the list.

C.9 Minimum Form Standard Pragmats.

Because some implementations may have special syntax for pragmats, portable programs should employ only minimum form pragmats:

pragmat-symbol, standard-item, pragmat-symbol.

where "standard-item" is PAGE, RES, UPPER, or POINT. Implementers should provide PRAGMATS OFF {R9.2} (and perhaps PRAGMATS ON) to control interpretation of pragmats.

C.1Ø "PORTCHECK" Option.

Despite good intentions, a programmer may violate portability rules by inadvertently employing a local extension. To guard against this, each implementation should provide a PORTCHECK pragmat option. While this option is in force, the compiler prints a message for each construct that it recognizes as violating some portability constraint.

}