CONTROL DATA

CONTROL DATA B.V.

ALGOL 68
VERSION I
REFERENCE MANUAL

CONTROL DATA®
CYBER 170 SERIES
CYBER 70 SERIES MODELS 72, 73, 74
6000 SERIES COMPUTER SYSTEMS

CONTROL DATA

CONTROL DATA B.V.

ALGOL 68
VERSION I
REFERENCE MANUAL

CONTROL DATA®
CYBER 170 SERIES
CYBER 70 SERIES MODELS 72, 73, 74
6000 SERIES COMPUTER SYSTEMS

REVISION RECORD	
REVISION	DESCRIPTION
A	Original printing.
(9-17-75)	
B	This revision reflects compiler level 1.1.
(9-17-76)	Changes and corrections result from product
	development and document evaluation.

Additional copies of this manual may be obtained from :

CONTROL DATA SERVICES B.V.
J.C. van Markenlaan 5
P.O. Box 111
RIJSWIJK (Z.H.)
THE NETHERLANDS.

Address comments concerning this manual to :

CONTROL DATA SERVICES B.V.
J.C. van Markenlaan 5
P.O. Box 111
RIJSWIJK (Z.H.)
THE NETHERLANDS.

or use Comment Sheet in the back of this manual.

CONTENTS

0. INTRODUCTION.

This reference manual includes information to prepare, compile and execute programs written in ALGOL 68 for the CONTROL DATA CYBER, 170 series, CYBER 70 series models 72,73,74, and 6000 series.
The language implemented is described in :

> Revised Report on the Algorithmic Language ALGOL 68, A. van Wijngaarden et al.(Eds), Mathematical Centre Tracts 50, Mathematisch Centrum, Amsterdam, 1976.

References to this document are made by writing the mnemonic RR, if necessary followed by a section number, e.g. RR.1.1.1.a.
Syntactical constructions ("hypernotions") are bracketed by < and > , e.g. < quote symbol > . In the sequel the mnemonic RM is used to refer to Record Manager.

Chapter 1 presents an overview of the ALGOL 68 system.
Chapter 2 and 3 include information to prepare programs.
Chapter 2 deals with deviations from RR; chapter 3 summarizes implementation-dependant aspects of the language as they are realized on the CONTROL DATA computer systems.
Compilation and execution of programs is described in chapter 4 and 5, respectively.
Chapter 6,7 and 8 are useful for those who want to improve program efficiency.
Some sample jobs are given in chapter 9.

1. ALGOL 68 SYSTEM DESCRIPTION.

1.1. COMPILER FEATURES.

- Virtually the full language, defined in RR is implemented, including e.g. parallel processing and binary, formatted and unformatted transput.

- Routines may be separately compiled and loaded with a particular program only when explicitly used in that program.

- User preludes and postludes may be precompiled to be used in subsequent compilation of programs, separate routines or again preludes and postludes. In the last case a new prelude and postlude is generated combining (in a nested fashion) the definitions in both preludes and postludes.

- Operators may be defined in terms of machine-instructions by means of intermediate code file macros (ICF macros). Thus defined operators are compiled into in-line code rather than calls to routines. ICF macros have been used to define almost all operators in the Control Data standard prelude.

- In case of errors in the source text, self-explanatory error messages are put out, including line number and severity of the error (warning, error, fatal).

- When during the syntactical scan of the source program an error is detected, error-recovery procedures are invoked that guarantee a syntactically correct intermediate output of that pass.

For a following pass then it is possible to detect more errors and to transform the erroneous program in an executable, even meaningful one. Optionally, programs containing no compilation errors of severity fatal may be executed.

- A source listing is provided, giving on each line the line number, the source text, the level of parenthesis nesting and the type of the last object on the line.
  (comment, string, tag, mode-indication, operator, denotation)

- Optionally, an assembly-like listing of the object program is provided.

- The allocation of data is designed in such a way that scope checking can be restricted to the checking of routines being called.

- The internal representation of internal objects has been chosen in such a way that values of mode reference-to-MODE or row-of-MODE use recognizable bit-patterns and thus no templates are needed, allowing a fast and simple garbage collection.

- Some enquiries have been added to the standard set, mainly for memory allocation statistics.

- In case of runtime errors, a self-explanatory diagnostic message, including the line number and module name, is printed.
  An error trace back of active routines at the time of interruption will be given.

- Optionally, a post-mortem symbolic dump will display all identifiers in all ranges active at the time of the interruption.

1.2. COMPILER STRUCTURE.

The compiler consists of a main overlay or CRADLE and six overlays (PASS 1 through PASS 6), each of which performs a complete pass over the program. Communication between the overlays is accomplished by means of :

- intermediate files IL1, IL2, IL3, IL4 (ICF) and IL5 (CF), each representing some intermediate form of the program ;

- the SYMBOL-TABLE, a comglomerate of all internal tables, such as Name-table, Identifier-table, etc. ;

- common blocks in the CRADLE.

CRADLE:

Controls loading of overlays, contains service routines and common blocks.
I/O routines are included in the overlays rather than in the CRADLE, because different passes require different I/O (RM) routines.

PASS 1 :

Lexical scan.
Performs control-card interpretation and SYMBOL-TABLE initialization.
A SYMBOL-TABLE overlay representing the prelude selected on the control-card is loaded.

Pass 1 reads the source text of the user program and produces IL1. Pass 1 creates SYMBOL-TABLE entries for all identifiers, fields, mode-indications, operators and most denotations, replacing these in the IL1 by SYMBOL-TABLE pointers. It translates the hardware representation into internal values. It determines all defining occurences of mode-indications and operators and performs preliminary parenthesis analysis.
Pass 1 also produces the source listing.

PASS 2 :

Mode independent parse.
Pass 2 reads IL1 and produces IL2. The parser has been generated by program from a context-free grammar.
IL2 represents the program in reversed Polish notation.
Appropriate entries for all modes are created in the SYMBOL-TABLE.

PASS 3 :

Mode-table cleanup, coercion and balancing. These functions are combined in one overlay, although they are logically independent.

1. Mode-table cleanup.
   This module does not scan the program but operates on the SYMBOL-TABLE only. It performs mode equivalencing and the ordering of the constituent modes of the union.

2. Coercion and balancing.
   Reads IL2 and produces IL3.
   IL2 is read backwards and IL3 represents the program in Polish notation.
   This module performs :
   - Identifier identification
   - Operator identification
   - Determination of all coercions and balancing of modes
   - The coercions are inserted in the IL3 produced.

PASS 4 :

Code-generator 1.
Pass 4 reads IL3 backwards (i.e. in the order of the source code) and produces IL4. IL4 is a modification of ICF, the input to the code-generator 2 phase of the SYMPL compiler.
IL4 is similar to machine code, but it contains no register indications.
The operands of instructions generated may be SYMBOL-TABLE pointers or pointers to preceding ICF instructions.

PASS 5 :

Code-generator 2.
Reads IL4 and produces IL5 (or CF).
Pass 5 is a slightly modified version of code-generator 2 of the SYMPL compiler.
It inserts load instructions if necessary and performs register assignment and instruction scheduling.

PASS 6 :

Editor.
Produces error messages from the error-entries in the SYMBOL-TABLE, created in earlier passes.
Reads IL5 (or CF) and produces standard SCOPE relocatable binary.
Optionally, it provides an edited object-code listing.

1.3. RUNTIME LIBRARY.

The runtime library is an integral part of the ALGOL 68 compiler system. This library contains a large number of routines, needed to support different features of the language.

1. Storage allocation.
   Garbage collector
   Core request routine

2. Language constructs.
   Parallel processing
   Generators involving multiples
   Assignations involving multiples
   Slicing
   Selections involving multiples
   Subscripting
   Row displays involving miltiples
   Coercions yielding multiples

3. Standard prelude.
   Standard environment enquiries
   Control Data additional environment enquiries
   Transput (coded in ALGOL 68)
   Basic I/O to support transput

4. Diagnostic system.
   Error trace back
   Post-mortem symbolic dump.

1.4. OPERATING SYSTEM INTERFACE.

The compiler operates as a user program to the operating system (SCOPE 3.4 or KRONOS 2.1).
It uses RM for compiler I/O exclusively.
Input and Output have FO = SQ, RT = Z.
Intermediate files have FO = WA, RT = F or RT = W.
The LGO-file has FO = SQ, RT = U.
System requests are used to load overlays, obtain the clock reading, etc.

1.5. HARDWARE CONFIGURATION.

The basic hardware configuration required for compilation consists of the minimum configuration required by the operating system.

2. DEVIATIONS FROM THE REVISED REPORT ON ALGOL 68.

2.1. SYNTACTICAL / SEMANTICAL DEVIATIONS.

1. No transient names.
A name referring to an element of a flexible row V is treated as a name referring to an element of a fixed multiple and does not change or lose its meaning when an assignment is made to the name referring to V. Hence the concept of transient name is abolished.
E.g. after

```
FLEX [1:4]  INT fri:=  (1,2,3,4);
REF INT i = fri [2] ;
REF REF INT ii = fri [2] ;
REF  [ ]INT il = fri [2:4] ;
fri:=  (7,8);
```

the following clauses always yield TRUE

```
i = 2;
REF INT (ii) IS i;
il [i]  IS i  .
```

2. No ghost element.
A multiple being an element of a flexible multiple is flexible. Thus the following declarations are equivalent:

```
FLEX [1:3]  FLEX [4:5]  INT ffi;
```

and

```
FLEX [1:3]       [4:5]  INT ffi;
```

and the following is legal

ffi [2] = (7,8,9) .

This obviates the need for a ghost element.

3. Scope of generators.
The scope of all generators, including those implied in variable declarations, is global.

4. Scope checking.
The scope of a value of mode PROCEDURE is not checked upon assignation or when yielded by a range or routine, but at the time the routine is called.

5. No VOID in united modes.
VOID is not allowed as constituent of a united mode; EMPTY can not be used as a void-denotation.

6. Bold symbols.
The CONTROL DATA implementation accepts "bold letter ABC symbols" and "bold DIGIT symbols" in operators and mode indications (see Table 1 and also section 3.1.3). These symbols are produced by the following additional rules:

```
< TAB > :: < BOL > .
< TAD > :: < BOL > .
< TAM > :: < BOL > .
< BOL > :: <bold letter ABC > ;
           <BOL, bold letter ABC> ;
           <BOL, bold DIGIT> .
```

2.2 DEVIATIONS IN HARDWARE REPRESENTATION.

A <style i sub symbol> has the following representation (expressed in ASCII-graphics, see Table 1):

(/

; a <style i bus symbol> has the following representation:

/)

2.3. DEVIATIONS IN STANDARD ENVIRONMENT.

In this section ALGOL 68 features are summarized which are not supported by the CONTROL DATA implementation.
For extra (non-ALGOL 68) features, one is referred to chapter 3, section 4 and 5.

2.3.1. CONTROL DATA PRELUDES AND POSTLUDES.

This implementation does not follow the concept of particular-prelude and particular-postlude (see RR.10.1).
Instead, this implementation assumes the existence of a "standard-postlude" and a "library-postlude" in analogy to the concept of standard-prelude and library-prelude, respectively.

The CONTROL DATA standard prelude will be available in three versions:

1. single precision version: all standard operators on operands having a size long-LONGSETY are omitted;

2. semantic long version: all standard operators on operands having mode REAL and COMPL with size long are included;

3. syntactic long version: the intention of this standard-prelude is to allow the use of algorithms containing modes with size long-LONGSETY in a single precision mode (this version is not implemented yet).

2.3.2. TRANSPUT DECLARATIONS.

-- Format pattern:
Format patterns (RR.10.3.4.9) are not supported by this implementation.

-- Staticizing of formats:
The entire format is "staticized" at the moment of attachment to the file.

-- Logical end while writing:
For "sequential access books" a write action sets "logical end of book" on the character with highest position in the line to be written.

-- Associate:
The routine "associate" is not supported by this implementation.

-- Conversion key:
The programmer can not provide his own conversion key; calling of the routine "make conv" (see RR.10.3.1.3.j) results in program termination.

3. CONTROL DATA ALGOL 68 LANGUAGE

3.0. INTRODUCTION.

This chapter deals with:

-- specification of what is not defined or partly defined in RR; what is left to the discretion of the implementor or can not be specified by RR alone, e.g. "undefined";

-- specific properties of the CONTROL DATA implementation, e.g. "environment enquiries";

-- extra (non-ALGOL 68, nonstandard) features.

3.1. HARDWARE REPRESENTATION.

3.1.1. INTERNAL REPRESENTATION OF SYMBOLS.

Internally the compiler uses a 12-bit code, called IL0-code, that meets the following requirements:

-- IL0-code is a superset of ASCII;

-- there is a one-to-one-mapping of all single-position (non-composite) representations of symbols mentioned in RR.9.4.1. into the class of 8-bit IL0-code items;

-- all 8-bit IL0-code items outside this mapping have a map in a "typographical display feature" or in a <style TALLY monad symbol> or in a <bold LETTER symbol> or in a <bold style i LETTER symbol> or in a <bold DIGIT symbol> or in a <style i LETTER symbol> ;

-- the rest of the IL0-code items are mapped into <other string item> .

A table giving for each IL0-code item the corresponding ALGOL 68 item, is presented in Table 1.

3.1.2. EXTERNAL REPRESENTATION OF SYMBOLS.

In table 1 relationships between the SCOPE 3.4/KRONOS 2.1 standard 63-character set, an ASCII subset and a subset of ALGOL 68 symbols are given.

3.1.2.1. ALGOL 68 SINGLE-POSITION REPRESENTATIONS.

The ASCII subset corresponding with the SCOPE 3.4/KRONOS 2.1 standard character set (see column 4 of Table 1) does not contain all single-position representations for symbols listed in RR.9.4.1.
Items occurring in both ASCII and ALGOL 68 representations are <style TALLY monad symbols> .

3.1.2.2. ALGOL 68 MULTIPLE-POSITION REPRESENTATIONS.

Symbols from ALGOL 68 having a representation which is a sequence of capital letters (e.g. mode standards, bold symbols, see RR.9.4.1) are represented by:

<bold glyph> <letter sequence>

in flagged mode, or

<bold glyph> <letter sequence> <bold glyph>

in stropped mode, where <bold glyph> is an IL0-item having code 27 (see Table 1, column 3), and <letter sequence> is the ASCII-code equivalent of the sequence of capital letters (see sections 4.1.2 and 4.3, also).
All other multiple-position representations are represented by their equivalent ASCII sequences.

3.1.2.3. GENERAL IMAGES.

Throughout a program one can represent ALGOL 68 symbols, given in Table 1 column 5, by their corresponding "general images". A general image corresponding to an ALGOL 68 symbol S, has in stropped mode the following format:

<bold glyph> <open symbol> <N> <close symbol> <bold glyph>

where <bold glyph> is as specified before, <open symbol> and <close symbol> are ALGOL 68 symbols, and <N> is the decimal representation of the IL0-code item corresponding to S, see Table 1 column 3.
In stropped mode general images of a string of ALGOL 68 symbols S1...Sk may be represented by:

<bold glyph> <open symbol> <N1,...,Nk> <close symbol> <bold glyph>

where N1,...Nk are the decimal representations of the IL0-code items corresponding to S1,...,Sk , respectively.

In flagged mode the representations are the same as the representations in stropped mode without the last <bold glyph> . In flagged mode one can not represent a general image close behind a flagged sequence of letters, e.g., expressed in ASCII-graphics:

```
--  in stropped mode'A(65)P' is equivalent to 'AAP'
    and '(65,66,67)' is equivalent to 'ABC' ;

--  'BIN(126) in flagged mode is equivalent to 'BIN'
    (126) in stropped mode;
```

-- 'BIN(126) in flagged mode is equivalent to 'BIN' 'SKIP' in stropped mode.

3.1.3. EXTERNAL REPRESENTATION OF MODE INDICATIONS AND OPERATORS.

Mode indications and operators are represented by:

< bold glyph > < TAG >

in flagged mode, or by

< bold glyph > < TAG > < bold glyph >

in stropped mode (see sections 4.1.2 and 4.3, also), where < bold glyph > is as specified before.

Another way to represent mode indications and operators is to use the (non-ALGOL 68, CONTROL DATA) bold symbols according to the rules in section 2.1.

3.2. PRAGMATS.

Pragmats are available now for:

-- Manipulation of source program input and output (see sections 4.1 and 4.2):

list-pragmat and nolist-pragmat;
eject-pragmat;
flagged-pragmat and stropped-pragmat;
state-pragmat and nostate-pragmat;
stop-pragmat.

-- Addition of (inline ICF) code to CONTROL DATA prelude, postlude and standard library (see section 4.1), and interfacing with other programming languages (see section 8) :

xdef-pragmat and fedx-pragmat;
xref-pragmat;
prog-pragmat;
inline-pragmat.

3.3. UNDEFINED.

This section specifies what actions are taken in "undefined" situations. For situations which lead to an error message, one is referred to section 5.2.

1. If the union or the integer in a conformity-clause or case-clause has the value SKIP or is not initialized, the out-CASE-clause is elaborated.

2. All values are initialized to SKIP.

3. Assignation to SKIP or NIL and dereferencing of NIL or SKIP will cause the program to be terminated and an error message to be put out.

4. Array bound violations in subscripting, slicing, assignation of multiple values or rowing of multiple values to multiple values of higher dimension will cause the program to be terminated and an error message to be put out.

5. The use of infinite or indefinite real values causes the program to be terminated and an error message to be put out.

6. An attempt to backspace a file with c OF cpos equal to 1 will not change cpos.

7. All undefined transput situations, except the one mentioned above, lead to an error message.

3.4. CONTROL DATA STANDARD PRELUDE.

3.4.1. STANDARD ENVIRONMENT ENQUIRIES.

The CONTROL DATA standard prelude contains the following standard enquiries (see RR.10.2.1):

```
INT    int lengths      = 1,
INT    int shorths      = 1,
INT    int width        = 15,
INT    max int          = 2**48-1,
INT    real lengths     = 1,
INT    real shorths     = 1,
INT    real width       = 16,
REAL   max real         = 2**1022*(2**48-1),
REAL   small real       = 2**(-47),
INT    exp width        = 3,
INT    bits lengths     = 1,
INT    bits shorts      = 1,
INT    bits width       = 48,
INT    bytes lengths    = 1,
INT    bytes shorths    = 1,
INT    bytes width      = 4,
INT    max abs char     = 4095,
CHAR   null character   = REPR 0,
CHAR   flip             = ≠T≠,
CHAR   flop             = ≠F≠,
CHAR   errorchar        = ≠*≠,
CHAR   blank            = ≠ ≠.
```

Note: the above given denotations are represented by CDC-graphics, see Table 1.

3.4.2. CONTROL DATA ADDITIONAL ENVIRONMENT ENQUIRIES.

In the CONTROL DATA standard prelude the mode "LONGBYTES" is declared as :

```
MODE LONGBYTES = STRUCT(BYTES b1, b2).
```

In addition to those specified in section 3.4.1, the CONTROL DATA standard prelude includes the following environment enquiries:

```
INT
    progsize              = C size of the program in
                            words C,
    maxprog               = C the maximum storage in
                            words available to this
                            job C,
    compiler level        = xxxyyzzz
                            CO xxx is the compiler level,
                               yy is the year of creation,
                               zzz is the day of creation.
                            CO,
PROC INT
    stacksize             = C size of the currently active
                            stack in words C,
    allocated stack       = C size of the currently allocat-
                            ed stack in words C,
    max allocated stack   = C maximum of allocated stack in
                            this program execution C,
    heapsize              = C size of the heap in words C,
    available             = C number of free words C,
    time limit            = C CP time in seconds allowed
                            for this job C,
```

```
PROC REAL
    clock                   = C CP time in seconds used in
                              this job C,
PROC LONGBYTES
    wall clock              = C time in the format hh.mm.ss,
                              hh is hours,
                              mm is minutes,
                              ss is seconds C,
    date                    = C date in the format dd.mm.yy,
                              dd is the day,
                              mm is the month,
                              yy is the year C,
INT
    collections             :=0 CO one is added for each gar-
                                   bage collector execution
                                CO,
    garbage                 :=0 CO the garbage collector adds
                                   the number of words of gar-
                                   bage faced every time it is
                                   executed
                                CO,
REAL
    collect seconds         :=0 CO garbage collector adds the
                                   CP time spend for every
                                   execution
                                CO.
```

3.4.3. CONTROL DATA ADDITIONAL OPERATORS AND ROUTINES.

One operator is added to the standard environment:

```
OP MODIV = (REF INT i, INT k)INT:
           (INT r= i+k; INT s= i-k*r; i:= r; s).
```

A different version of MODIV is */:= .

Three routines are added to the standard environment:

```
PROC(INT)INT memory=
           C if the parameter value is larger than the
             current field length, then increase the
             field length to the parameter value with-
             out, however, exceeding the maximum field
             length C,
PROC VOID collect gargabe =
           C invoke the garbage collector C,
PROC VOID error =
           C causes normal termination; triggers
             trace-back and symbolic dump C.
```

3.5. TRANSPUT DECLARATIONS.

Transput is performed via Record Manager.

3.5.1. CHANNELS.

The channels included in the CONTROL DATA standard environment are given in the following table.

channel name	re-set	set	get	put	bin	compr.	maxpos (page, line char.)	standconv
standin channel	f	f	t	f	f	f	(1,max,80)	Display Code
standback channel	t	t	t	t	t	f	(1,1,max)	none
standout channel	f	f	f	t	f	t	(max,60,136)	Display Code
ztype channel	t	f	t	t	f	f	(max,max,35000)	Display Code
binarysequential channel	t	f	t	t	t	f	(max,max,3500)	none
charcompress channel	t	f	t	t	f	t	(max,max,17500)	none
charnoncompress channel	t	f	t	t	f	f	(max,max,17500)	none
ascii channel (£)	t	f	t	t	f	t		8-bit ASCII

(£) not implemented

Notes :

max = 131071,f=false,t=true;
reidf is false on all channels;
estab is true on all channels.

3.5.2. MAPPING OF TRANSPUT CONCEPTS INTO RM CONCEPTS.

The following table shows the mapping of ALGOL 68 Transput concepts into RM concepts.

ALGOL 68 entity	RM entity
book.	sequential (SQ) file, except for a book of standback channel, which is mapped on word addressable (WA) file.
logical end of a book.	End-of-Information (EOI).
idf of a book.	logical file name.
line.	RM record having record type (RT): Z for standin channel, standout channel, ztype channel and ascii channel; W for binary sequential channel, char compress channel and char noncompress channel; F for standback channel.
page end.	End-of-Section (EOS), only for channels having lines mapped on RM records with RT = W.
open.	RM open, the book must be in chainbfile.
establish.	RM open, the book should not be in chainbfile, but may exist in the job.
lock.	RM close.
scratch.	RM close and unload.

Note, that because "idf of book" is identical to the "logical file name" of the operating system, one is subject to the restrictions for logical file names. The ALGOL 68 system does not support the use of the FILE control card.

3.5.3. THE H-PATTERN.

The H-pattern (not defined in RR) may occur at positions where a <literal> is legal, see RR.10.3.4.1.i.

An H-pattern is defined as

<letter h symbol, enclosed clause> ,

where the enclosed clause yields a <strong row of character NEST coercee> .
The H-pattern is a means to specify dynamically a row-of-character in a format in positions where a literal is allowed.

3.5.4. BINARY TRANSPUT.

The routines putbin (RR.10.3.6.1.a) and getbin (RR.10.3.6.2.a) will accept parameters of the mode procedure-with-reference-to-file-parameter-yielding-void.

4. COMPILATION

Compilation can be directed by control card options and pragmats.

Several pragmats are available to control compiler input and output and source text interpretation. They will be described in the appropriate sections of this chapter.

4.1. PRAGMATS FOR INPUT TO COMPILATION.

4.1.1. END OF SOURCE TEXT.

The end of the text of a particular-program to be compiled or a routine to be seperately compiled is marked by the pragmat PR stop PR.
However, the end of the last particular-program or routine in a sequence of compilations should be marked with an End-of-Record (EOR).
This pragmat is useful when the text to be compiled is interactively given to the compiler via a connected input file.

4.1.2. STROPPED MODE AND FLAGGED MODE.

Stropped mode and flagged mode can be controlled by the use of pragmats (see section 4.3, also).
The pragmat PR flagged PR causes switching to flagged mode.
The pragmat PR stropped PR causes switching to stropped mode.

4.1.3. SEPARATE COMPILATION.

A routine text may be separately compiled by enclosing its unit in the pragmats PR naming xdef PR and PR fedx PR, where "naming" is the entry-point name that will be generated for the routine.

The text to be separately compiled consists of one or more identity-declarations or operation-declarations with a routine-text on the right-hand side, the unit of which is surrounded by the above pragmats.
The last declaration is followed by SKIP.

An operator or identifier having a mode PROCEDURE may be defined as external by a declaration of the form:

```
OP operator = formal part :
PR xref naming PR SKIP
```

or

```
PROC identifier = formal part :
PR xref naming PR SKIP,
```

respectively. Here "formal part" consists of formal-parameters, if any, followed by formal-declarer, while "naming" is the entry point for the routine (see section 9, sample 1).
If the routine is entered via an interface routine, the format of the xref-pragmat is different (see section 8, no.2).

There are some restrictions for the name naming:

- a semicolon in naming will be skipped;
- a colon in naming will cause an error message, but the colon will appear in the entry name (such an entry name is illegal for the loader);
- all other symbols in naming are accepted, except those with ILO-code less or equal 31 (decimal) and greater or equal 96 (decimal) (see Table 1); these symbols are replaced by a period-symbol.

4.1.4. LIBRARY ADDITION.

The pragmat PR prog PR is used to add code to the library-prelude and the (non-ALGOL 68) library-postlude (see section 2.1, point 7).
The source text of the code to be added is given to the compiler, operating in "library addition mode" (see section 4.1, N-option), in the following format:

```
naming :
 <STYLE begin symbol>
prelude addition ;
PR prog PR
postlude addition
 <STYLE end symbol>  .
```

The compiler generates an overlay module in binary form, containing the CONTROL DATA prelude and postlude including the additional code.
A thus created overlay module, when placed in a library, can be used in subsequent compilations by referring to it with the name "naming" (see section 4.1, P-option).
The items "prelude addition" and "postlude addition" stand for serial-clauses to be added to the library-prelude and library-postlude, respectively.
Here are two examples:

```
example 1:    plot:
              ( PROC plot = C...C ;
              PR prog PR
              SKIP )      ;
```

example 2:

```
file :
BEGIN FILE nonstand; open(nonstand,...) ;
PR prog PR
stop: close(nonstand)
END
```

Example 2 redefines a.o. the label stop. Another example is given in section 9, sample 1.

4.1.5. INLINE OPERATORS.

Operators expressed in ICF macros can be compiled inline. Therefore the inline-pragmat must be used in the following way:

```
OP formal part     operator =
PR inline
icf-macros
PR SKIP
```

Here "formal part" stands for formal-parameters followed by formal-declarer, "operator" stands for a TAO-symbol, "icf-macros" is a sequence of ICF macros, while the other items stand for themselves.
For examples, see section 9, sample 1.

The use of ICF macros is described in the IMS (Internal Maintenance Specification) of this compiler.

4.2. PRAGMATS FOR OUTPUT FROM COMPILATION.

The following pragmats may be used to control the output of the source listing:

-- PR list PR, which turns on output of source listing;
PR nolist PR, which turns off output of source listing;

-- PR eject PR, which causes the line which contains this pragmat to be put out on the first line of the next page.

-- PR state PR, which turns on output of parenthesis level and construct type on the source listing;
PR nostate PR, which turns off output of parenthesis level and construct type on the source listing;
The S-option does not prevail over these two pragmats (see section 4.3).

The control card option L=0 prevails over the above pragmats (see section 4.3).

4.3. CONTROL CARD FOR COMPILATION.

The following control card formats can be used to invoke the ALGOL 68 compiler:

A68(p1,p2,...,pk) or A68,p1,p2,...,pk. ;

none of the parameters p1 through pk is mandatory.
A parameter is either of the format

XY=sequence or XY .

The various parameters are specified below.

-- I: source input:

omitted	: source input from file INPUT;
I	: source input from file COMPILE;
I=lfn	: source input from file "lfn".

-- C: the number of significant characters on source input line:

omitted	: 72 significant characters on source input line;
C	: 72 significant characters;
C=n	: "n" significant characters (n must be less or equal 130).

-- F: stropped mode and flagged mode (see section 3.1,also):

omitted	: the compiler operates in stropped mode, i.e. a "bold word" is represented by its characters, surrounded by < bold glyphs> ;
F	: the compiler operates in flagged mode, i.e. a bold word is represented by its

characters, preceded by a
< bold glyph >. Note, that a
bold word must be followed by a
character that can not occur in
a bold word.

-- B: (relocatable) binary output:

omitted	: binary output on file LGO;
B	: binary output on file LGO;
B=lfn	: binary output on file "lfn";
B=0	: binary output is suppressed.

-- L: source listing:

omitted	: source listing and diagnostics on file OUTPUT;
L	: source listing and diagnostics on file OUTPUT;
L=lfn	: source listing and diagnostics on file "lfn";
L=0	: diagnostics on file OUTPUT, source listing and object listing are suppressed.

-- S: suppression of parenthesis level and construct type:

omitted	: no suppression of parenthesis level and construct type;
S	: parenthesis level and construct type are suppressed.

-- O: object code listing in COMPASS-like form:

omitted	: no object code listing is produced;
O	: object code listing on file OUTPUT;
O=lfn	: object code listing on file "lfn";
O=0	: no object code listing is produced.

-- X: combined use of L-,S- and O-option (X stands for a combination of L,S and O):

omitted	: source listing and diagnostics on file OUTPUT; no suppression of parenthesis level and construct type;
X	: diagnostics on file OUTPUT; if X contains L, then source listing on file OUTPUT; if X contains S, then parenthesis level and construct type are suppressed; if X contains O, then object code listing on file OUTPUT;
X=lfn	: the same as X, but instead of file OUTPUT the file "lfn" is taken;
X=0	: diagnostics on file OUTPUT; source listing and object code listing are suppressed.

-- A: inline subscription:

omitted	: subscripting is done by a bound-checking runtime routine;
A	: subscripting is done by inline-code; no bound-checking is preformed.

-- P: overlay module containing prelude and postlude is taken from the runtime library:

omitted	: default module is taken from library A68LIB;
P	: default module is taken from library A68LIB;
P=naming	: module having name "naming" is taken from library A68LIB;
P=lib/naming	: module having name "naming" is taken from library named "lib";

P=0 : no prelude is used; the definitions of PLAIN modes are internally generated.

-- N: library addition mode (see section 4.1.4):
omitted : the compiler operates in normal mode;
N : the compiler operates in library addition mode.

-- D: a symbolic dump is produced at a runtime error:
omitted : no symbolic dump is produced;
D : a symbolic dump is produced.

-- T: object code is produced if there are no errors of severity fatal (see section 4.4):
omitted : if there are errors of severity E or F, no object code is produced;
T : if there are errors of severity F, no object code is produced.

-- Z: scheduling of instructions to reduce runtime:
omitted : no scheduling of instructions;
Z : the compiler does scheduling of instructions.

4.4. COMPILE-TIME DIAGNOSTICS.

Errors are detected in the following phases of the compiler: lexical scan, syntactical scan, mode equivalencing and mode checking.
The output of diagnostics is presented per phase, ordered by line-number within that phase; it is written behind the source listing.
There are three categories of severity for errors:

W : warning, object code is produced, if required (see section 4.3, B-option);

E : error, object code is produced optionally (see section 4.3, T-option);

F : fatal, no object code is produced.

Abortion of compilation can only be triggered in a phase after the syntactical scan.

4.5. OPERATION CHARACTERISTICS.

For a program consisting of ten lines of code, the compiler needs 52K when no "long modes" are specified.

5. EXECUTION.

5.1. CONTROL CARD FOR EXECUTION.

The control card for execution (program control card) has the following three options:

-- R: recovery from abnormal job termination (see SCOPE Reference manual, RECOVR function):

omitted	: recovery from abnormal conditions specified by 77 (octal);
R	: recovery from abnormal conditions specified by 77 (octal);
R= DD	: recovery from abnormal conditions specified by "DD" (octal);

-- M: specification of maximum field length:

omitted	: maximum field length is 131071 (decimal);
M	: maximum field length to be used is the current field length;
M= DDDDDD	: maximum field length is "DDDDDD" (octal).

-- I: specification of increment for memory request:

omitted	: increment is 2000 (octal);
I	: increment is 2000 (octal);
I= DDDDDD	: increment is "DDDDDD" (octal).

The ALGOL 68 system guarantees after each garbage collection a free space of the number of words specified by the I-option. If necessary, memory requests are issued up to the maximum, specified by the M-option.

5.2. OBJECT-TIME DIAGNOSTICS.

Upon detection of errors during program execution the ALGOL 68 run-time system will perform the following actions:

-- send an error message to the output file (see also section 3.4, L-option), specifying the type of error, erroneous data (if applicable), line number and relevant identifier;

-- give on the output file a trace-back of all active routines, optionally interspersed with a symbolic dump of all identifiers, except labels, contained in these routines;

-- terminate execution of the program.

The following errors are diagnosed:

-- procedures called out of scope;

-- invalid arguments of standard functions;

-- core overflow;

-- time-out;

-- undefined situations as described in section 3.3.

6. HINTS FOR EFFICIENCY.

-- Use identity-declaration for a value, not having a mode ROWS-of-MODE, whenever that value is used more than once.

-- Use contracted declarations to declare several names having the same mode.
E.g. :

```
        REAL x,y;
```

rather than

```
        REAL x, REAL y;
```

and

```
        [1:5]  INT x,y;
```

rather than

```
        [1:5] INT x,  [1:5]  INT y;
```

-- Write the actual declarer in the declaration rather than defining and using a mode indication.
E.g. :

```
        [1:7]  INT j1;
```

rather than

```
        MODE A = [1:7] INT; A j1;.
```

-- In defining the fields of a structured value, do begin with the fields having plain modes and place the fields having modes REF-to-MODE at the end. This will save some time during garbage collection.

-- Avoid accessing of identifiers defined in a routine, that is not the current routine, because indirect addressing is needed for such identifiers.

-- Use collateral elaboration.

-- In formula's, try to place function calls before identifiers in order to avoid storing of intermediate results.

-- Avoid rowing.
E.g. after [1:1] INT i; use

```
i[1] := 7;
```

rather than

```
i  := 7;.
```

-- Avoid slicing, if the resulting slice does not occupy consecutive storage locations.
The "holes" are not recovered by the garbage collector and assignation is faster for compact multiples.

-- Avoid staticizing the same format more than once.
Use the sequence

```
printf(format);
DO printf(data) OD;
```

rather than

```
DO printf((format, data)) OD; .
```

7. RUNTIME ORGANIZATION.

7.1. STORAGE ALLOCATION.

The storage allocated to a value always consists of a "static direct part"; depending on the mode of the value, a "static indirect part" or a "dynamic part" may be a constituent.
For a value having a mode PLAIN, PROCEDURE or UNITED there is only a static direct part; for a value of the mode REF-to-MODE, the reference is the static direct part, and the value referred to is the static indirect part.
For a value of the mode ROWS-of-MODE, the descriptor is the static direct part, the elements constitute the dynamic part.
For a value of the mode REF-to-ROWS-of-MODE, the reference is the static direct part, the descriptor is the static indirect part and the elements constitute the dynamic part.
In this implementation only the static direct part of identifiers and intermediate results are allocated on the stack.
The storage allocated to the static parts of all identifiers in a range is the SID or static identifier stack of that range and the storage used for the static parts of intermediate results is the SWO or static working stack of that range.
For each active range the stack contains a SID and a SWO.
For each active routine the stack contains the return-information, followed by the entries for the active ranges.

The outer range of a routine has a different type of header word, called procedure header; the SID of that range represents the actual parameters of the routine. The layout of the stack segment allocated for routine elaboration is completely determined at compile-time; all addressing in the stack is performed using fixed offsets relative to the procedure-header of the relevant routine-activation.

7.2. PARAMETER MECHANISM.

The call to a routine with parameters is carried out in the following steps :

1. elaborate the PLAIN to be called yielding a procedure-word (see section 7.3., no. 5) ;
2. place the procedure-word on top of the stack;
3. place a zero on top of the stack;
4. elaborate the parameters and place these on top of the stack;
5. call the runtime routine G;CALL with register X1 set to the address of the procedure-word in the stack;
   G;CALL sets the following registers :
   A1 : address of procedure-word;
   B7 : stack pointer of calling procedure (i.e. old B1);
   B1 : link-address field of procedure word (see 7.3.5);
   B5 : address of the entry point of the routine being called;
   B6 : copy of A1, i.e. address of the procedure word;
   and transfers control to the routine.
   Scope checking is also performed in G;CALL;
6. the routine called, when compiled by ALGOL 68, calls the runtime subroutine G;PROL with a parameter word (called routine-info) in register A1, X1. (for contents of routine-info, see section 7.3., no.9);
7. G;PROL replaces the procedure-word and the zero-word in the stack with the return-information and procedure header;
8. the routine-text is elaborated, where parameters are accessed just as normal identifiers;

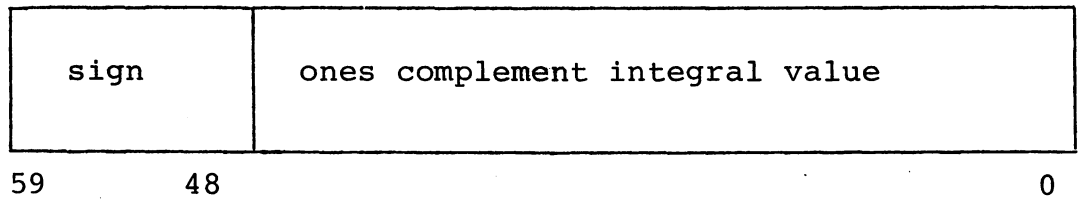
9. the value yielded by the routine is transmitted as follows :
   a. if it is a one word value: in register X6;
   b. if it is a two word value: in register X6 and X7;
   c. if it is a multiple word value: on the stack, where the first word overwrites the return-information (see section 7.3., no. 10);
10. using the return-information (fetched earlier in case c of step 9), control is transferred to the calling routine;
11. the calling routine resets stack-pointer register B1 to the procedure header of the calling routine by means of subtraction of a compiler determined constant offset.

7.3. REPRESENTATION OF INTERNAL OBJECTS.

As may be seen in the sequel, the internal representation of objects is such, that values of mode REF-to-MODE and ROW-of-MODE can be recognized and therefore no templates are needed for garbage collection.

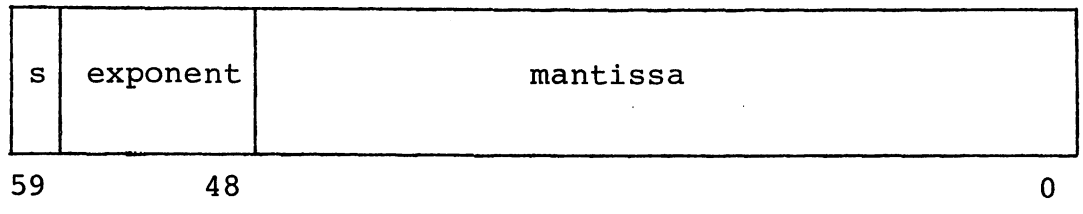
1. Non-long PLAIN modes.

INT:



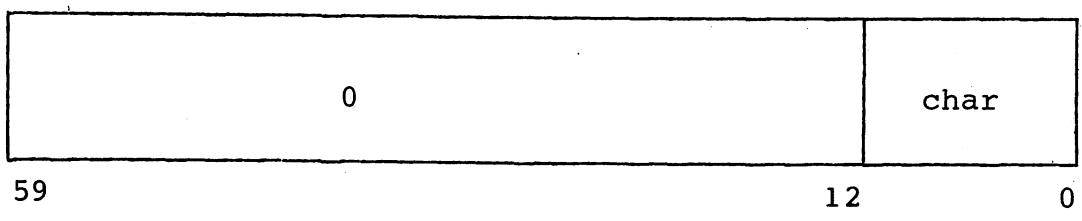
The sign bits are equal to bit 47, the sign of the integral value.

REAL:



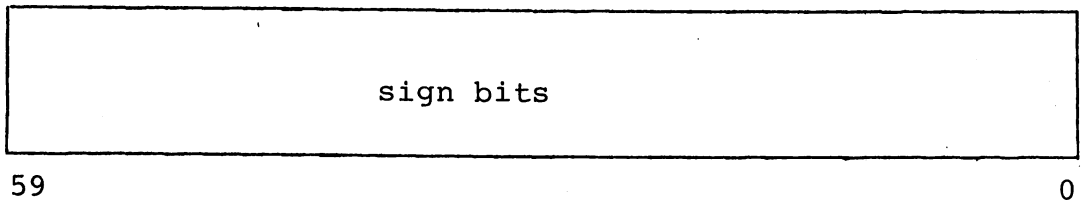
This is the standard floating point representation, where s is the sign of the mantissa (one bit), and the exponent is biased by 2000 (octal) and inverted for a negative mantissa.

CHAR:



Here char is the 12-bits character with the same representation as the internal representation in IL0 used in the compiler (see table 1, column 3).

BOOL:



When TRUE, sign bits consists of 60 one-bits.
When FALSE, sign bits consists of 60 zero-bits.

2. Long PLAIN modes.

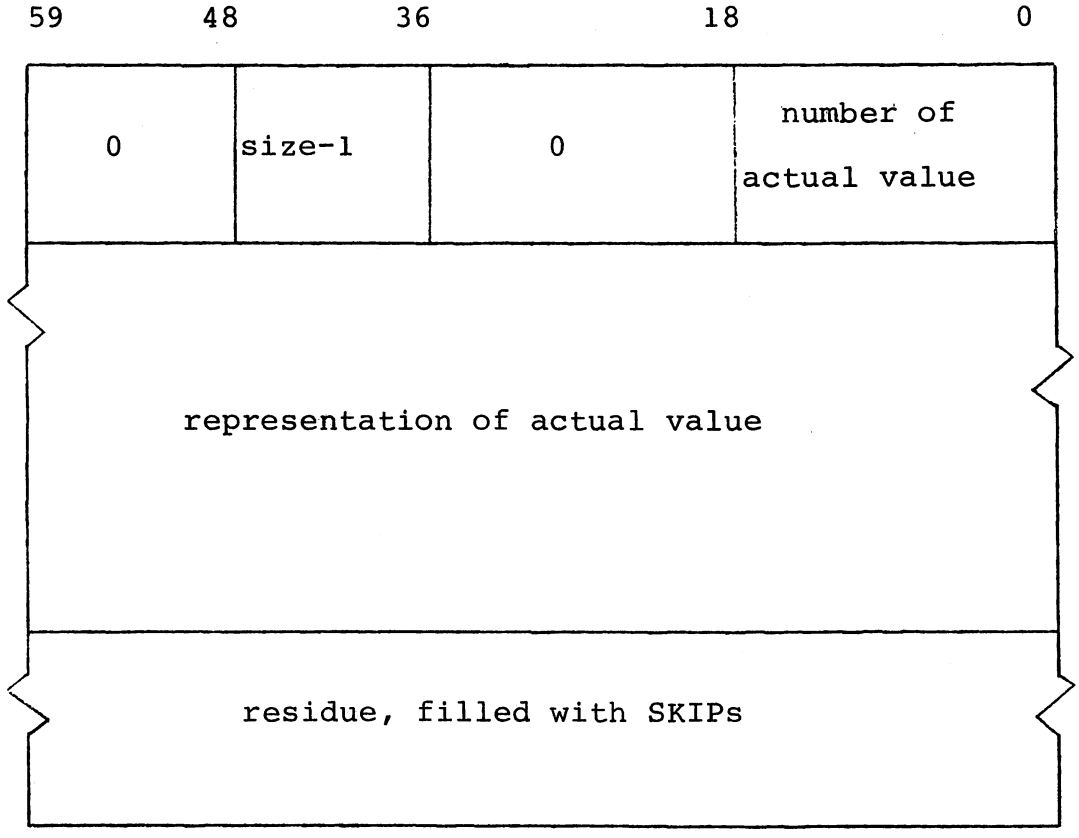
Values having a mode PLAIN with size long occupy two words in storage each of which has the format of the corresponding "non-long" PLAIN mode.
Only REAL with size long is supported.

3. Structures.

Values of mode STRUCT(MODE1 iden1,...,MODEn idenn) are represented by the concatenation of the internal representations of the fields iden1 through idenn of mode MODE1 through MODEn.

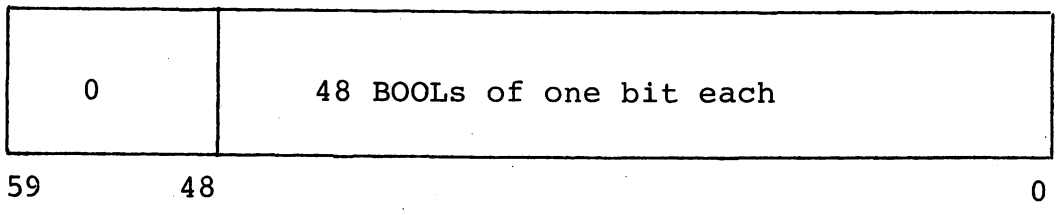
4. Unions.

Values of mode UNION(MODE1,MODE2,...,MODEn) occupy the number of words required for the longest object of the set of modes (MODE1, MODE2,...,MODEn), plus one word for the so called union-word :



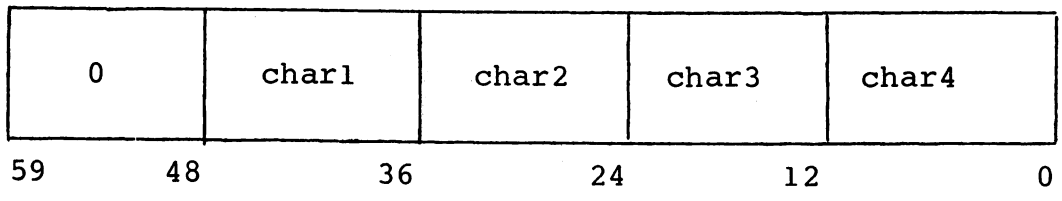
The first word in an internal representation of a union is the union-word, where size is the number of words occupied by this representation.

BITS:



The BOOLs are numbered from left to right :
when TRUE : 1;
when FALSE : 0.

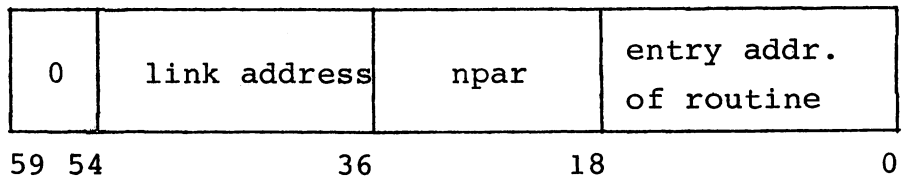
BYTES:



Here char1, char2, char3, and char4 are the "significant" bits of values of the mode CHAR; the characters are numbered from left to right.

5. ROUTINES.

a. A value of the mode PROCEDURE not requiring scope checking occupies one word in storage :



This word is called procedure-word (or proc-word). Depending on the type of routine represented, the link address-field is :

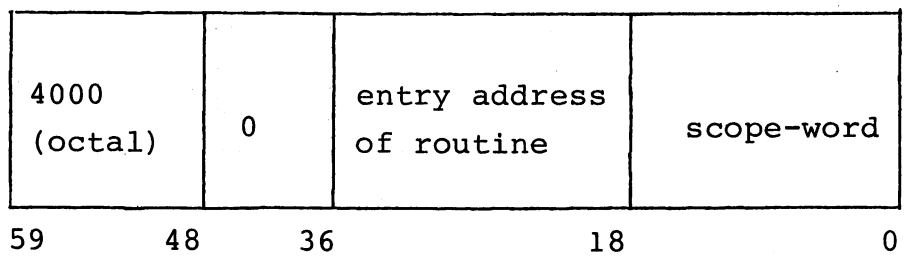
Routine text : the link address is the stack-pointer of the routine containing the range that is the scope of this routine; it is also called static link;

Label : the link address is the stack-pointer of the routine containing the label;

Other language routine : the link address is the entry address of the routine written in the other language; the routine is indirectly called via the interface routine.

npar is zero, except in the last case, where it is the number of parameters.

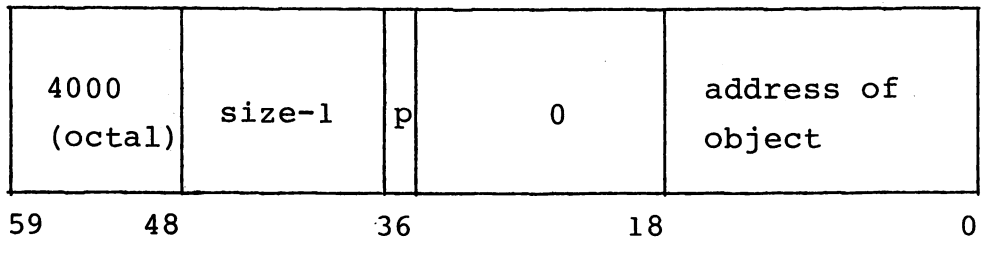
b. Values of the mode PROCEDURE that require scope checking are represented by the following word:



Scope-word is the address of a scope control word on the heap, generated at range-entry of the range having the scope of the routine (see also 12, range-header).

6. References.

Values of the mode REF-to-MODE occupy one word in storage :



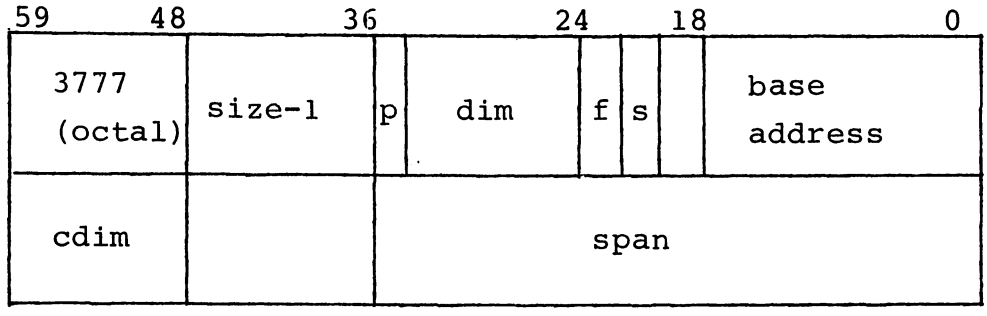
Here, size is the number of words allocated to the static indirect part, while :

p=1 : when the object contains references or multiples, i.e. must be traced in garbage collection;

p=0 : otherwise.

7. Multiples.

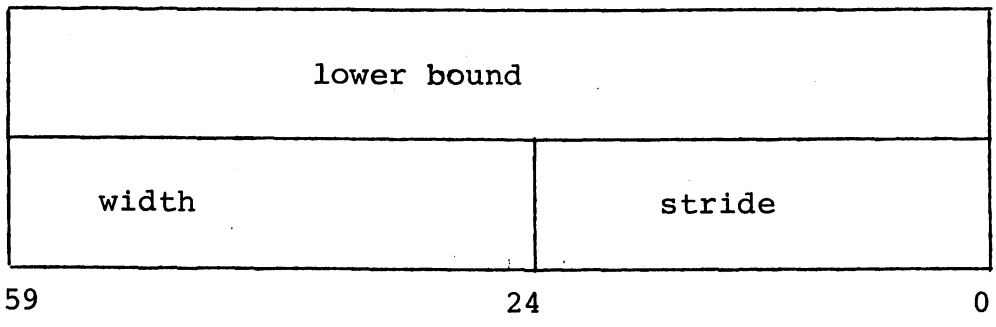
Values of the mode ROWS-of-MODE are represented by a descriptor, consisting of a two-word header and one two-word triple for each dimension, and by the elements.
The header has the following layout :



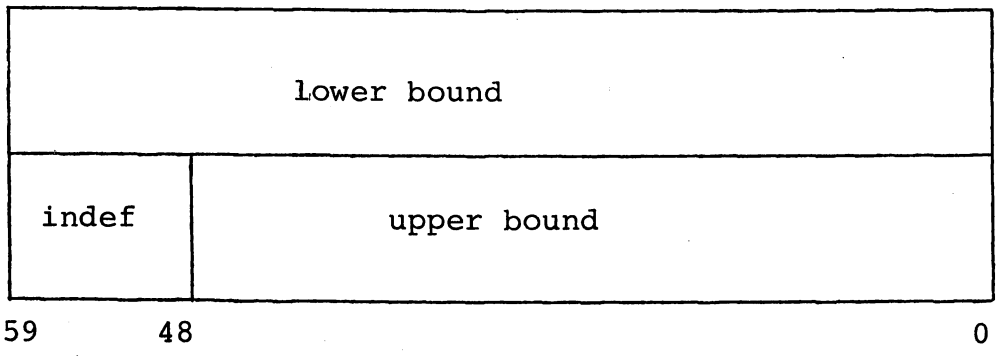
The items size and p are the same as specified under references, see 6. above. Further :

dim = the number of dimensions;
f = 1 for a flexible multiple, and
= 0 for a fixed multiple;
s = reserved;
base address = the address of the first element of the multiple;
cdim = a number 0 $\leq$ cdim $\leq$ dim, indicating the "compactness" of the multiple, viz. one less than the number of nested loops needed to access all elements of the multiple.

The layout of a triple is as follows.
For a non-empty multiple :



and for an empty multiple :



width = upper bound - lower bound + 1;
stride = the distance in memory locations of two elements, the subscripts of which differ by one in the dimension selected;
indef = an indefinite exponent :
1777(octal) for positive upper bound,
6000(octal) for negative upper bound.

8. MODULE DESCRIPTOR.

Each relocatable module generated contains a four-word area including data and parameters used mainly for object-time diagnostics.
The layout of this area is :

name of program	line table address
relative address of code in the module	start of module
0	address of range table
0	E;SYMDP

59 18 0

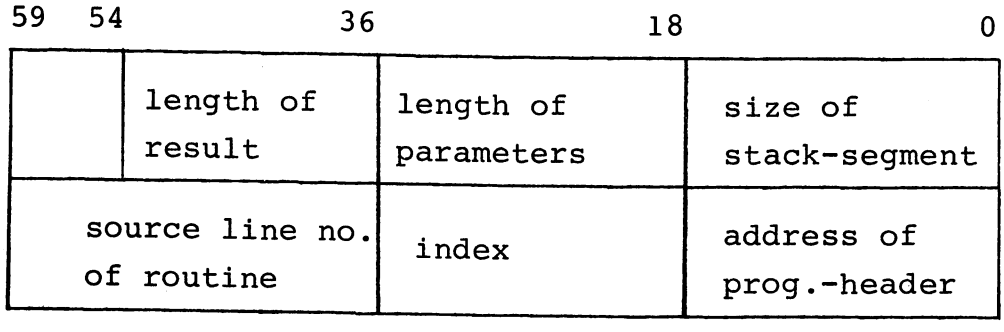
The meaning of the various fields is :

- name of module :
  a name consisting of seven Display Code characters;
- line table address :
  the line table serves to determine the line number of the source text from a relative address in the code;
- relative address of code in the program :
  is used to convert an absolute address into a relative address in the code;
- start of program :
  physical start of module;

- address of range table :
  the range table is used for symbolic dump only. It allows to determine the nesting of ranges and the identifiers defined within these ranges. This field is set to zero when the D-option (see section 4.3) is not selected;
- E;SYMDP :
  is the address of the runtime routine for a symbolic dump; this field is zero when the D-option (see section 4.3) is not selected. Thus, the symbolic-dump routine is not loaded when no module in the load is compiled with the D-option selected.

9. ROUTINE-INFO.

The routine-info (or procedure-info) is a two-word entry associated with a routine text:



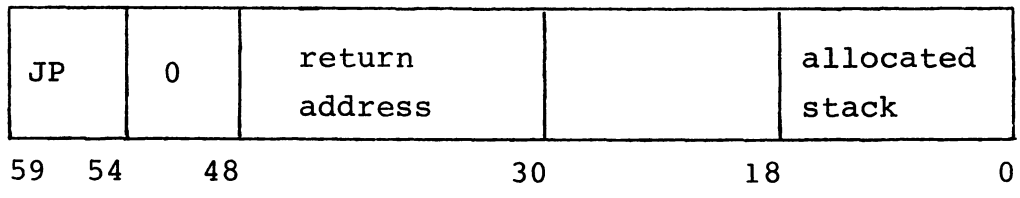
length of result = the length of the result;
length of parameters = the length of the parameters;
size of the stack-segment = the maximum length of the SWO plus the SID;
source line no. of routine = the line number in the source text where the routine is defined;

index = the index in the range-table for the range of the parameters of the routine;

address of program header = the address of the program header

10. RETURN-INFORMATION.

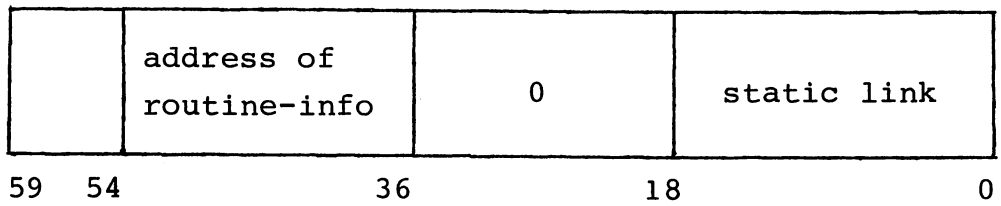
This is a word in the stack used to return from a routine. The subroutine G;PROL stores this word in the stack.



The left half (30 bits) is the jump-instruction generated in G;CALL by the hardware for the return-jump instruction.
The allocated stack field is the maximum allocated stack for its routine and all the dynamically surrounding routines.

11. PROCEDURE-HEADER.

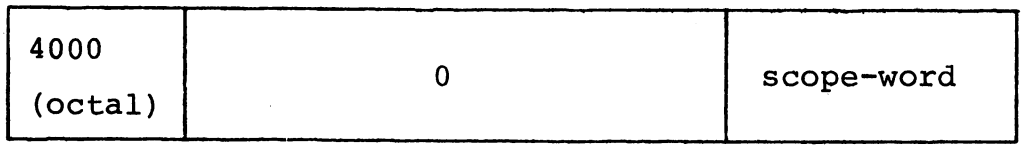
Each active routine has a procedure-header generated in G;PROL :



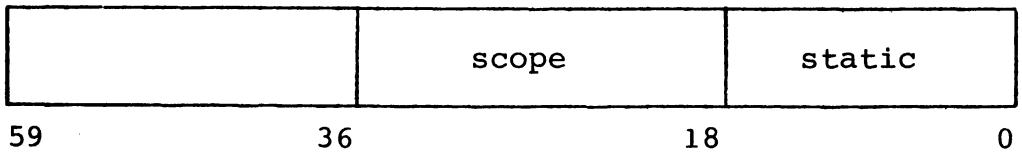
The routine-info is used as parameter to a runtime routine for exit from a routine yielding a result having a size long. It is also used for trace back and symbolic dump.
The static link field is used to access identifiers, operators and modes from surrounding routines.
For routines with global scope the static link field is set to zero, because some register is always pointing to the global stack-segment.

12. RANGE-HEADER.

For each active range being the scope of some routine a one word entry is placed in the stack.



The item scope-word is the address of a scope control word on the heap.
For each range-entry a unique word is provided in the format.



scope = the address of the range-header in the stack;
static = the address of the procedure-header of the routine containing the range.

8. INTERFACE WITH OTHER PROGRAMMING LANGUAGES.

Routines written in other languages may be called from ALGOL 68 in either of the following ways :

1. By defining an operator written in ICF (see section 1) which calls the routine directly (RJ-instruction).
   Parameters must be passed in the X-registers and ICF-instructions to load them must be explicitly stated in the operator definition.
   This method is normally reasonable only when the routine is written in COMPASS.

2. By defining an identifier or operator with an xref-pragmat.
   The routine is entered directly from G;CALL (see section 7.2.) or via an interface-routine.
   In the first case control is passed to the entry-point by a JP-instruction. The parameters can be found on the stack. For the contents of registers, see section 7.2.6. no. 6.
   If an interface-routine is used, both the entry-point of the interface-routine and the entry-point of the actual routine must be mentioned in the xref-pragmat. The interface-routine can find the address of the actual routine in B1 or in the procedure-word. The number of parameters can also be found in the procedure-word (see section 7.3. no.5.).
   If the registers B1, B2, B3, B4 are used by the called routine, they must be saved either by the actual routine or by the interface-routine.
   In the runtime library one interface-routine is provided for calling subroutines or functions

written in FORTRAN(FTN). This routine is called A68FTN. It saves the B-registers.
The correspondence between the ALGOL 68 parameters and the FORTRAN parameters is :

ALGOL 68	FORTRAN
REF INT	INTEGER
REF REAL	REAL
REF LONG REAL	DOUBLE PRECISION
REF COMPLEX	COMPLEX
REF LONG COMPLEX	array with 2 elements DOUBLE PRECISION
REF CHAR, REF BITS, REF BYTES	INTEGER

Other parameters are not allowed. If the FORTRAN routine is a FUNCTION, the result mode in ALGOL 68 and the function type in FORTRAN have the same relation as the parameters, except for the REF.
It is obvious that LONG COMPLEX is not possible as a result mode.
If the FORTRAN routine is a SUBROUTINE, the ALGOL 68 result mode is VOID.

A sample job is given in section 9.

9. SAMPLE JOBS.

Sample 1: library addition and seperate compilation.


```
0001,          VECTOR: ≡ EXAMPLE OF LIBRARY ADDITION ≡                                          0
0002,          ↑BEGIN↑                                                                          1
0003,            ↑OP↑ ([]↑REAL↑,[]↑REAL↑) ↑REAL↑ ↑* =                                           1    MONAD
0004,                 ≡ INNER PRODUCT, WRITTEN IN COMPASS, ADRESSES OF DESCRIPTORS ≡           1
0005,                 ≡ IN X2,X3, RESULT IN X6 ≡                                              1
0006,            ↑PR↑ INLINE                                                                  1    PRAGMAT
0007,              HEAD    4,1                                                                1    PRAGMAT
0008,              SRV     /1,2                                                               1    PRAGMAT
0009,              SRV     /2,3                                                               1    PRAGMAT
0010,              PCAL    $VECPRO                                                            1    PRAGMAT
0011,          R1  DRV     6                                                                  1    PRAGMAT
0012,            ↑PR↑ ↑SKIP↑,                                                                 1
0013,            ↑OP↑ * = ([]↑REAL↑ A,B) []↑REAL↑:      ≡ VECTOR MULTIPLICATION ≡             1
0014,                 ↑PR↑ XREF VECMUL ↑PR↑ ↑SKIP↑,                                           1
0015,                 ≡ OPERATOR FOR CONVERSION OF 32 BITS TWO COMPLEMENTS ≡                  1
0016,                 ≡ NUMBERS (IN ↑BITS↑) TO ↑INT↑ AND V.V. ≡                               1
0017,            ↑OP↑ (↑BITS↑) ↑INT↑ ↑B2I↑ =                                                  1
0018,            ↑PR↑ INLINE                                                                  1    PRAGMAT
0019,              HEAD    2,1                                                                1    PRAGMAT
0020,              LXOR    /1,=16R80000000=                                                   1    PRAGMAT
0021,          R1  ISUB    *-1,=16R80000000=                                                  1    PRAGMAT
0022,            ↑PR↑ ↑SKIP↑,                                                                 1
0023,            ↑OP↑(↑INT↑)↑BITS↑ ↑I2B↑ =                                                    1
0024,            ↑PR↑ INLINE                                                                  1    PRAGMAT
0025,              HEAD    2,1                                                                1    PRAGMAT
0026,              IADD    /1,=16R80000000=                                                   1    PRAGMAT
0027,          R1  LXOR    *-1,=16R80000000=                                                  1    PRAGMAT
0028,          ↑PR↑ ↑SKIP↑,                                                                 1
0029,            ↑PRIO↑ ↑* = 6,                                                               1
0030,            ↑PROC↑ FUNC  = (↑REF↑ ↑REAL↑ X) ↑REAL↑:       ≡ FORTRAN FUNCTION ≡           1
0031,                 ↑PR↑ XREF A68FTN,FUN ↑PR↑ ↑SKIP↑;                                       1
0032,            ↑PR↑ PROG ↑PR↑   ↑SKIP↑                                                      1
0033,          ↑END↑                                                                          0
PROGRAM LENGTH     000023B WORDS
REQUIRED  CM 052300. CP    .561 SEC.
SPECIFIED OPTIONS           NZ
```

```
0001,                ≡ EXAMPLE OF SEPARATE COMPILATION ≡                                                    0
0002,                                                                                                       0
0003,            (                                                                                          1
0004,              †OP† * = ([]†REAL† A,B) []†REAL†:                                                        1
0005,              †PR† XDEF VECMUL †PR†                                                                    1
0006,                (                                                                                      2
0007,                   []†REAL† X = A[†AT† 1], Y = B[†AT† 1];                                              2
0008,                   †INT† N = †UPB† X, M = †UPB† Y;                                                     2
0009,                   †IF† N /= M                                                                         3        TAG
0010,                   †THEN† PRINT((NEWLINE,≠VECTORS NOT MATCHING, LENGTHS ARE≠,                          6
0011,                              N,M,NEWLINE));                                                           4
0012,                         STOP                                                                         4        TAG
0013,                   †FI†;                                                                               2
0014,                   [N]†REAL† S;                                                                        2
0015,                   †FOR† I †TO† N †DO† S[I]  := ([I]*Y[I] †OD†;                                        2
0016,                   S                                                                                   2        TAG
0017,                ) †PR† FEDX †PR†;                                                                      1
0018,              †SKIP†                                                                                   1
0019,            )                                                                                          0
PROGRAM LENGTH     000263B WORDS
REQUIRED  CM 052500. CP    .935 SEC.
SPECIFIED OPTIONS           AZ
```

```
   FUNCTION FUN        74/74   OPT=1                FTN 4.5+410        15/09/76  10.29.54        PAGE     1

 1                 FUNCTION FUN(X)
               C COMPUTE EXP(X)
                   F=X
                   S=X+1.0
 5                 DO 10 I=2,10
                   F=F*X/I
                   S=S+F
                10 CONTINUE
                   FUN=S
10                 RETURN
                   END
```

```
LIBRARY(LIB,NEW)
ADD(*,LGO)
FINISH.
ENDRUN.
```

```
0001.          TEST:                                                                          0
0002.          (      ↑INT↑ N;                                                                1
0003.                 READ(N);                                                                1
0004.                 [1:N]↑REAL↑ A,B;                                                        1
0005.                 READ((A,B));                                                            1
0006.                 PRINT((≠A≠,NEWLINE,A,NEWLINE,≠B≠,NEWLINE,B,NEWLINE,                     3
0007.                       ≠A*B≠,NEWLINE,A*B,NEWLINE,≠A↑*B≠,NEWLINE,A↑*B,NEWLINE));        1
0008.                 PRINT((≠      INTEGER                  2-COMPLEMENT≠                    3   STRING
0009.                       ≠                                 1-COMPLEMENT≠,NEWLINE));        1
0010.                 PRINTF($G,2X2R32D,2X2R48DL$);                                           1
0011.                 ↑FOR↑ J ↑TO↑ 10                                                         1   DENOT.
0012.                 ↑DO↑                                                                    2
0013.                    ↑INT↑ I = (J v-(2**31),1-2**31, -4,-2,-1,                            4
0014.                              0,1,2,4,2**31-1);                                          2
0015.                    ↑BITS↑ K = ↑I2B↑ I;                                                  2
0016.                    PRINTF((I,K,↑BIN↑ ↑B2I↑K))                                           2
0017.                 ↑OD↑;                                                                   1
0018.                 PRINT((≠           X                    FUN(X)≠                      3   STRING
0019.                       ≠                 EXP(X)≠,NEWLINE));                            1
0020.                 ↑FOR↑ I ↑TO↑ N↑OVER↑2                                                   1   DENOT.
0021.                    ↑DO↑ PRINT((A[I],FUNC(A[I]),EXP(A[I]),NEWLINE)) ↑OD↑                 1
0022.          )                                                                              0
PROGRAM LENGTH     0010118 WORDS
REQUIRED  CM 053400. CP  1.571 SEC.
SPECIFIED OPTIONS          P
```

```
A
+1.0000000000000000E  +0 +2.0000000000000000E  +0 +3.0000000000000000E  +0 -1.0000000000000000E  +0 -2.0000000000000000E  +0
-3.0000000000000000E  +0 +1.0000000000000000E  +0 +2.0000000000000000E  +0 +3.0000000000000000E  +0 -1.0000000000000000E  +0
-2.0000000000000000E  +0 -3.0000000000000000E  +0
B
+4.0000000000000000E  +0 +5.0000000000000000E  +0 +6.0000000000000000E  +0 -4.0000000000000000E  +0 -5.0000000000000000E  +0
-6.0000000000000000E  +0 -4.0000000000000000E  +0 -5.0000000000000000E  +0 -6.0000000000000000E  +0 +4.0000000000000000E  +0
+5.0000000000000000E  +0 +6.0000000000000000E  +0
A*B
+4.0000000000000000E  +0 +1.0000000000000000E  +1 +1.8000000000000000E  +1 +4.0000000000000000E  +0 +1.0000000000000000E  +1
+1.8000000000000000E  +1 -4.0000000000000000E  +0 -1.0000000000000000E  +1 -1.8000000000000000E  +1 -4.0000000000000000E  +0
-1.0000000000000000E  +1 -1.8000000000000000E  +1
A+*B
+0.0000000000000000E  +0
```

INTEGER	2-COMPLEMENT	1-COMPLEMENT
-2147483648	10000000000000000000000000000000	11111111111111111111111111111111
-2147483647	10000000000000000000000000000001	11111111111111111111111111111110
-4	11111111111111111111111111111100	11111111111111111111111111111011
-2	11111111111111111111111111111110	11111111111111111111111111111101
-1	11111111111111111111111111111111	11111111111111111111111111111110
+0	00000000000000000000000000000000	00000000000000000000000000000000
+1	00000000000000000000000000000001	00000000000000000000000000000001
+2	00000000000000000000000000000010	00000000000000000000000000000010
+4	00000000000000000000000000000100	00000000000000000000000000000100
+2147483647	01111111111111111111111111111111	00000000000000000000000000000000

```
          X                    FUN(X)                    EXP(X)
+1.0000000000000000E  +0 +2.7182818011463330E  +0 +2.7182818284590410E  +0
+2.0000000000000000E  +0 +7.3889947089946020E  +0 +7.3890560989306100E  +0
+3.0000000000000000E  +0 +2.0079665178570960E  +1 +2.0085536923187650E  +1
-1.0000000000000000E  +0 +3.6787946428570710E  -1 +3.6787944117143920E  -1
-2.0000000000000000E  +0 +1.3537918871252330E  -1 +1.3533528323661330E  -1
-3.0000000000000000E  +0 +5.3325892857150280E  -2 +4.9787068367863840E  -2
```

```
 RCN  REKENCENTRUM PETTEN  3.4.3/406 R.D 09.17
10.28.17.CDA687K  FROM    /CD
10.28.17.IP  00001152 WORDS - FILE INPUT  , DC 00
10.28.17.CDA68,CM60000,T40.
10.28.24.ACCOUNT,019100,90,900.
10.28.24.REDUCE.
10.28.25.ATTACH,A68,ID=CDA68,MR=1.
10.28.25.PFN IS
10.28.25.A68
10.28.25.PF CYCLE NO. = 001
10.28.25.ATTACH,A68LIB,ID=CDA68,MR=1.
10.28.25.PFN IS
10.28.25.A68LIB
10.28.25.PF CYCLE NO. = 001
10.29.26.LOCKIN.
10.29.26.COMPASS(L=0,S=A68LIB/A68TEXT)
10.29.27.  ASSEMBLY COMPLETE.   36000B  CM USED.
10.29.27.      0.302  CPU SECONDS  ASSEMBLY TIME.
10.29.27.RFL(55000)
10.29.27.A68(N,Z)
10.29.41.   -VECTOR -      NO ERRORS
10.29.41.  CM 052300. CP    .561 SEC.
10.29.41.A68(A,Z)
10.29.48.   -VECMUL -      NO ERRORS
10.29.48.  CM 052500. CP    .935 SEC.
10.29.53.REDUCE.
10.29.53.FTN(R=0)
10.29.59.       .087 CP SECONDS COMPILATION TIME
10.29.59.EDITLIB(L=DUMMY)
10.30.12.SKIPR(INPUT)
10.30.12.COPYBR(INPUT,X)
10.30.12.REWIND(X)
10.30.13.COPYSBF(X)
10.30.15.RETURN,LGO.
10.30.15.RFL(55000)
10.30.19.A68(P=LIB/VECTOR)
10.30.34.   -TEST   -      NO ERRORS
10.30.34.  CM 053400. CP   1.571 SEC.
10.30.34.REDUCE.
10.30.34.LGO.
10.30.41.TEST     - A68 1.1.0 76236
10.30.44.STOP-     0.396 CP SECONDS USED
10.30.44.OP  00002048 WORDS - FILE OUTPUT , DC 40
10.30.44.CP        6.612 SEC.         7 ADJ. SEC.
10.30.44.IO        3.732 SEC.
10.30.44.CS       80.317 KWS.         4 ADJ. SEC.
10.30.44.SS                           11
10.30.44.PP    -   31.974 SEC.      DATE 15/09/76
10.30.44.MFL SINCE LAST SUMMARY  055000B
10.30.44.EJ  END OF JOB, CD
```

Sample 2: Ackermann function.

```
                                * SOURCE LISTING *          A68 1.1.0 76256      15/09/76  11.48.17.     PAGE       1


   0001.          ACKERMANN:  ≡ CALCULATES ACKERMANN(3,N) ≡                                                    0
   0002.          (                                                                                            1
   0003.            ↑PROC↑ ACKERMANN = (↑INT↑ M,N) ↑INT↑:                                                      1
   0004.               ↑IF↑ M = 0 ↑THEN↑ N+1                                                                  3     DENOT.
   0005.               ↑ELIF↑ N=0 ↑THEN↑ ACKERMANN(M-1,1)                                                     4
   0006.               ↑ELSE↑ ACKERMANN(M-1,ACKERMANN(M,N-1))                                                 4
   0007.               ↑FI↑.                                                                                  1
   0008.            ↑REAL↑ T1,T2, ↑INT↑ J, K:=64, K1:=16;                                                     1
   0009.            PRINT((≠  M,ACKERMANN(3,M),TOTAL TIME,TIME PER CALL,MAX STACK SIZE≠,                      3
   0010.                  NEWLINE));                                                                          1
   0011.            PRINTF ($2ZD,X13ZD,2X2ZD.3D2X,3X5ZD.D3X, 3X6ZDL$ );                                       1
   0012.            ↑FOR↑ I ↑FROM↑ 3 ↑TO↑ 8                                                                   1     DENOT.
   0013.            ↑DO↑                                                                                      2
   0014.               T1:=CLOCK;                                                                             2
   0015.               J:= ACKERMANN(3,I);                                                                    2
   0016.               T2:=CLOCK;                                                                             2
   0017.               PRINTF((I,J,T2-T1,3000000*(T2-T1)/(512*K1-15*K+9*I+37),                                4
   0018.                     MAX ALLOCATED STACK));                                                           2
   0019.               (K*:=2,K1*:=4)                                                                         2
   0020.            ↑OD↑                                                                                      1
   0021.          )                                                                                           0
PROGRAM LENGTH     000616B WORDS
REQUIRED  CM 053000. CP  1.800 SEC.
```

M,ACKERMANN(3,M),TOTAL TIME,TIME PER CALL,MAX STACK SIZE				
3	61	0.485	35.0	509
4	125	3.387	37.5	957
5	253	1.627	38.3	1853
6	509	6.574	38.2	3645
7	1021	26.276	37.9	7229
8	2045	105.145	37.7	14397

Sample 3: program with errors.


```
0001.          (                                                                    1
0002.          ↑REAL↑ X = 3.14,Y;                                                   1
0003.          ↑INT↑ I,J,X; J:=Y;                                                   1
0004.          ↑OP↑ ↑P↑ = (↑INT↑ K,J) ↑VOID↑:                                       1
0005.            ↑IF↑ I=K ↑THEN↑ ↑FOR↑ M ↑TO↑ M ↑DO↑ J:=J+1; ↑FI↑                    1
0006.          I ↑P↑ X;                                                             1
0007.          PRINTF($3ZD.3D↑3D$,I,J);                                             1
0008.          ↑MODE↑ ↑A↑= ↑STRUCT↑(↑REF↑ ↑B↑ A),                                   1
0009.                 ↑B↑= ↑STRUCT↑(↑REF↑ ↑A↑ A),                                   1
0010.                 ↑C↑= ↑UNION↑(↑A↑,↑B↑);                                        1
0011.          )                                                                    0

ERRORS DETECTED DURING THE LEXICAL SCAN

***      5 E     ↑OD↑ SYMBOL MISSING
***      5 E     ↑FI↑ SYMBOL MISSING
***      5 E     ↑IF↑ SYMBOL FOUND MATCHING THE CURRENT CLOSING SYMBOL, BUT CONTENTS  NOT YET FINISHED
***      7 E     + FORMAT ERROR: INCONSISTENT FRAME IN PICTURE
***      7 E     D FORMAT ERROR: REPLICATOR HAS NO ARGUMENT
***      7 E     D FORMAT ERROR: INCONSISTENT FRAME IN PICTURE

ERRORS DETECTED DURING THE SYNTAX CHECKING

***      2 F     SYNTAX ERROR  170  : IDENTIFIED-AS SYMBOL MISSING IN IDENTITY DECLARATION
                     =  ↑CONSERR INSERTED BEFORE ;
***      5 E     SYNTAX ERROR   25  : NO DECLARATION, LABEL OR UNIT FOLLOWING SEPARATOR IN SERIAL CLAUSE
                      ↑EMPTY↑ ↑OD↑ INSERTED BEFORE ↑FI↑
***      5 E     SYNTAX ERROR   49  : UNIT NOT FOLLOWED BY SEPARATOR
                     ; INSERTED BEFORE I
***      6 E     ↑P↑ IS USED AS DYADIC OPERATOR WITHOUT PRIORITY
                     HIGHEST PRIORITY IS ASSUMED
***     10 E     SYNTAX ERROR   25  : NO DECLARATION, LABEL OR UNIT FOLLOWING SEPARATOR IN SERIAL CLAUSE
                      ↑EMPTY↑ INSERTED BEFORE )

ERRORS DETECTED DURING THE MODE EQUIVALENCING

***     10 E     UNION WITH ONLY EQUIVALENT MEMBER

ERRORS DETECTED DURING THE MODE CHECKING

***      2 F     MULTIPLE DECLARARATION FOR X IN THIS RANGE.
***      3 F     MULTIPLE DECLARARATION FOR X IN THIS RANGE.
***      3 F     MODE-ERROR IN ASSIGNATION
***      5 F     IDENTIFIER M HAS NOT BEEN DECLARED
***      5 F     MODE-ERROR IN ASSIGNATION
***      7 F     CALL TO PROCEDURE WITH WRONG NB OF ARGUMENTS
REQUIRED  CM 052100. CP   .373 SEC.
```

10. TABLES

T A B L E .

SCOPE 3.4/KRONOS standard 63-char. set item		IL0 - code (ASCII) item		ALGOL 68 item
Display Code (octal)	CDC-graphic	code (hexa-decimal)	ASCII-graphic	
		00-1F	control	typographical display feature
55	blank	20	blank	space symbol
		21	!	style TALLY monad symbol
64	≠	22	"(quote)	quote symbol
60	≡	23	##	style two comment symbol
53	$	24	$	formatter symbol
	(£)	25	%	percent symbol
67	∧	26	&	ampersand symbol
70	↑	27	´(apostr.)	bold glyph
51	(	28	(	open symbol, brief begin symbol
52	)	29	)	close symbol, brief end symbol
47	✻	2A	✻(aster.)	asterisk symbol
45	+	2B	+	plus symbol
56	,	2C	,(comma)	comma symbol
46	-	2D	-	minus symbol
57	.	2E	.(period)	point symbol
50	/	2F	/	divided by symbol
33-44	0 - 9	30-39	0 - 9	digit zero symbol - digit nine symbol
63	: (££)	3A	:(colon)	colon symbol, up to symbol, label symbol, routine symbol

(£) In CDC 64-graphic set, Display Code 63 is the percent.

(££) In CDC 64-graphic set, Display Code 00 is the colon.

T A B L E . 1 (continued)

SCOPE 3.4/KRONOS standard 63-char. set item		IL0 - code (ASCII) item		ALGOL 68 item
Display Code (octal)	CDC-raphic	code (hexa-decimal)	ASCII-graphic	
77	;	3B	;(semicol.)	go on symbol
72	<	3C	<	less than symbol
54	=	3D	=	equals symbol, is defined as symbol
73	>	3E	>	greater than symbol
71	↓	3F	?	style TALLY monad symbol
74	≤	40	@	at symbol
01-04	A - D	41-44	A - D	letter a symbol - letter d symbol
05	E	45	E	letter e symbol, times ten to the power symbol
06-32	F - Z	47-5A	F - Z	letter f symbol - letter z symbol
61	[	5B	[	brief sub symbol
75	≥	5C	\	times ten to the power symbol
62	]	5D	]	brief bus symbol
76	¬	5E	^(circumfl.)	style TALLY monad symbol
65	↑	5F	_(underl.)	style TALLY monad symbol
		60		style TALLY monad symbol
		61-64		style one letter a symbol - style one letter d symbol
		65		style one letter e symbol, times ten to the power symbol

T A B L E . 1 (continued)

SCOPE 3.4/KRONOS standard 63-char. set item		IL0 - code (ASCII) item		ALGOL 68 item
Display Code (octal)	CDC-graphic	code (hexa-decimal)	ASCII-graphic	
		66-7A		style one letter f symbol - style one letter z symbol
		7B		style TALLY monad symbol
66	V	7C	|	brief then symbol, brief else symbol, brief in symbol, brief out symbol
		7D		style TALLY monad symbol
		7E		skip symbol, tilde symbol
		7F	DEL	typographical display feature
		80		of symbol
		81		times ten to the power symbol
		82		differs from symbol
		83		up symbol
		84		down symbol
		85		floor symbol
		86		ceiling symbol
		87		and symbol
		88		or symbol
		89		times symbol

T A B L E . 1 (continued)

SCOPE 3.4/KRONOS standard 63-char. set item		IL0 - code (ASCII) item		ALGOL 68 item
Display Code (octal)	CDC-graphic	code (hexa-decimal)	ASCII-graphic	
		8A		over symbol
		8B		plus i times symbol
		8C		nil symbol
		8D		brief comment symbol
		8E		is at most symbol
		90		is at least symbol
		91		not symbol
		92-AF		style TALLY monad symbol
		B0-B9	(£)	bold digit zero symbol - bold digit nine symbol
		BA-C0		style TALLY monad symbol
		C1-DA	(£)	bold letter a symbol - bold letter z symbol
		DB-E0		style TALLY monad symbol
		E1-FA	(£)	bold style one letter a symbol - bold style one letter z symbol
		FB-FF		style TALLY monad symbol
		100-FFF		other string item

(£) Non-ALGOL 68 symbols.

INDEX

INDEX

C O M M E N T S H E E T

==========

J.C. van Markenlaan 5
P.O. Box 111
RIJSWIJK (Z.H.)
THE NETHERLANDS.

TITLE : ALGOL 68 version 1 Reference Manual

PRODUCT NO. : HOLHNL - 01 REVISION B

This form is not intended to be used as an order blank. Control Data Services b.v. solicits your comments about this manual with a view to improving its usefulness in later editions.

Applications for which you use this manual.

Do you find it adequate for your purpose?

What improvements to this manual do you recommend to better serve your purpose?

Note specific errors discovered (please include page number reference).

General comments :

FROM name : ..... position .....
company name : .....
address : .....