stichting
mathematisch
centrum

MC

DEPARTMENT OF COMPUTER SCIENCE IW 32/76 JUNE

L. AMMERAAL

# MINI ALGOL 68 USER'S GUIDE

First printing March 1975
Second edition June 1976

2e boerhaavestraat 49 amsterdam

*Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.*

AMS(MOS) subject classification scheme (1970): 68A00, 68A30

ACM -Computing Reviews- category: 4.12

# Mini ALGOL 68
# User's Guide

by

L. Ammeraal

ABSTRACT

Mini ALGOL 68 is a simple and compact sublanguage of ALGOL 68. Its syntax is given by a syntactic chart. A Mini ALGOL 68 compiler was written for the CONTROL DATA 6000 and CYBER machines. It is described how this compiler can be used.

KEY WORDS & PHRASES: *ALGOL 68, sublanguage, compiler*

CONTENTS

## 1. INTRODUCTION

Mini ALGOL 68 is a sublanguage of ALGOL 68 ([1],[2]). In spite of its compactness, Mini ALGOL 68 is rather powerful compared with some more established programming languages. Compared with ALGOL 68, an implementation of Mini ALGOL 68 is likely to require considerably less storage space and it may therefore be more appropriate for certain applications like time-sharing.

In the past eight months a compiler and a run-time-system for Mini ALGOL 68 were written by the author of this report.

The compiler was first written in ALGOL 60, and then, for reasons of efficiency, rewritten in an ALGOL 60-like subset of PASCAL. It produces assembler code for the CONTROL DATA 6000 and CYBER machines. Some care has been taken, however, to facilitate modifications, if wanted, for the generation of code for other machines.

## 2. DESCRIPTION OF THE LANGUAGE

To describe a complete new language, both its syntax and its semantics have to be defined. For a sublanguage of an already defined language, however, it is sufficient to define its syntax, since the semantics of all constructs that are allowed by this syntax are the same as for the original language.

For a user's guide, a context-free-grammar complemented with some informal remarks seems to be the best choice, both to aim at a certain degree of preciseness and to keep it well-readable. It should be kept in mind, however, that for ALGOL-like languages a context-free-grammar alone may not exclude certain invalid constructs like, e.g.,

```
    begin int i; real i; i := true end.
```

The grammatical remedy for this is to use a Van Wijngaarden grammar rather than a context-free-grammar, but for didactic reasons this method is not employed here.

For users of Mini ALGOL 68 who are already familiar with ALGOL 68, the difference between these languages can roughly be indicated by mentioning the absence of the ALGOL 68 concepts *structured values, united modes, heap generators, mode declarations, operator declarations, casts, flexible bounds, trimming, formats, jumps, completers, parallel processing.* The reader who is not familiar with these concepts but is interested in them is referred to the ALGOL 68 report ([1],[2]), or to an informal introduction to this language ([3],[4]). Rather than focussing the attention to the lacking ALGOL 68 concepts, a more positive way of introducing Mini ALGOL 68 could be followed as well. The fact that Mini ALGOL 68 is a small language is significant not only for the implementer but also from an educational point of view. The language can easily be learned. If it turns out to be insufficient for certain applications, the user needs only extend his knowledge to the full ALGOL 68 language or some larger sub-language, rather than learning a complete new language.

Since this user's guide is not intended as a tutorial textbook the syntax of Mini ALGOL 68 is given here in a rather formal way. On the other hand, it seems useful to present the grammar in a more visual way than in a list of production rules.

Therefore this grammar is given here by means of a syntactic chart. To define a notion, a diagram for that notion is given consisting of boxes $\boxed{\quad\quad}$ and nodes ⬭ connected by arcs. As a heading, each diagram has a number written inside a triangle with the abbreviated name of the notion to its right, e.g. "/6\ loop {5}".

In this example, 6 is the number of the diagram defining a *loop*, just below the heading, and 5 is the number of a diagram containing a reference to this definition. This means that diagram 5 contains at least one box of the form $\boxed{\text{loop}\ _{6}}$. Here the number 6 in the right bottom corner of the box refers to the defining diagram number 6 for *loop*.

In a diagram one easily generates a production of a notion by following a path, starting left at the top and ending right at the bottom. Boxes and nodes are passed from left to right unless otherwise indicated by arrows.

In a box another notion is given, whereas a node contains a (terminal) symbol. The rule for the insertion of comments is given here, rather than complicating the syntactic diagrams. Any sequence of characters, with the exception of #, preceded and followed by the character #, is allowed anywhere in the program without altering its meaning. Blank spaces outside strings have no meaning. As to style alternatives, a symbol listed in the left hand column below may be replaced by the corresponding symbol in the right hand column, without any restrictions.

| | |
|---|---|
| begin<br>if<br>case<br>[ | ( |
| end<br>fi<br>esac<br>] | ) |
| then<br>else<br>in<br>out | \| (at a 711 display terminal typed as an exclamation mark "!") |
| elif<br>ouse | \|: |

The underlining of symbols such as in begin is used here, but these symbols are actually enclosed by a pair of apostrophes like in 'BEGIN' for the use of punched cards or of other restrictive hardware. For typographical reasons most notions are abbreviated in the syntactic chart according to the following list which includes the diagram numbers. (Operators and the notion "chr" for an arbitrary character are defined more informally at the end (of the chart, though with the same referencing method as used for the diagrams.)

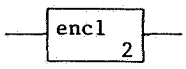
## ABBREVIATIONS AND DIAGRAM NUMBERS

| | | |
|---|---|---|
| ass | (15) | assignation |
| call | (33) | call |
| casecl | (35) | case clause |
| chclb | (37) | chooser clause using boolean |
| chcli | (38) | chooser clause using integral |
| chr | (48) | character |
| clcl | (3) | closed clause |
| collcl | (36) | collateral clause |
| condcl | (34) | conditional clause |
| dcla | (10) | actual declarer |
| dclf | (11) | formal declarer |
| dclv | (12) | virtual declarer |
| decln | (14) | declaration |
| denb | (27) | boolean denotation |
| denc | (30) | character denotation |
| deni | (28) | integral denotation |
| denot | (19) | denoter |
| denr | (29) | real denotation |
| dens | (31) | string denotation |
| digit | (26) | digit |
| dtor | (13) | declarator |
| encl | (2) | enclosed clause |
| form | (39) | formula |
| idtf | (24) | identifier |
| letter | (25) | letter |
| locgen | (17) | local generator |
| loop | (6) | loop |
| midn | (20) | mode identity declaration |
| mvdn | (21) | mode variable declaration |
| opd1 | (40) | priority 1 operand |

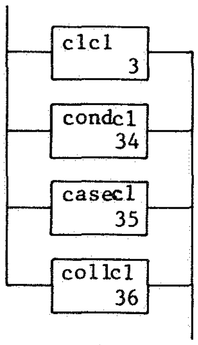
| | | |
|---|---|---|
| opd 2 | (41) | priority 2 operand |
| ' | ' | ' ' |
| ' | ' | ' ' |
| ' | ' | ' ' |
| opd8 | (47) | priority 8 operand |
| optor1 | (49) | priority 1 operator |
| optor2 | (50) | priority 2 operator |
| ' | ' | ' ' |
| ' | ' | ' ' |
| ' | ' | ' ' |
| optor8 | (56) | priority 8 operator |
| optorm | (57) | monadic operator |
| pidn | (22) | procedure identity declaration |
| planf | (8) | formal plan |
| planv | (9) | virtual plan |
| prim | (18) | primary |
| progr | (1) | program |
| pvdn | (23) | procedure variable declaration |
| rtext | (7) | routine text |
| scl | (4) | serial clause |
| slice | (32) | slice |
| tert | (16) | tertiary |
| unit | (5) | unit |

## Syntactic chart for Mini ALGOL 68

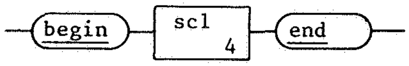
1 progr



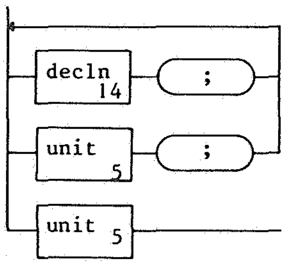
2 encl {1,18}



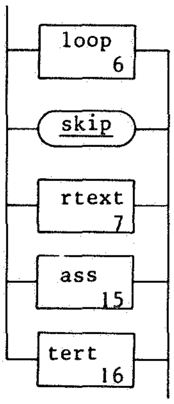
3 clcl {2}



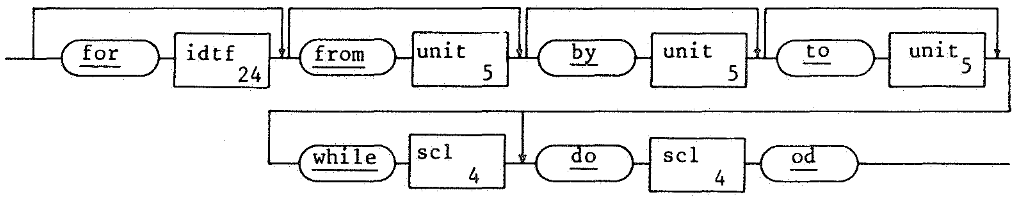
4 scl {3,6,33,34,35,37,38}



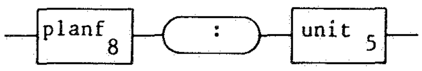
5 unit {4,6,7,10,15,20,21,32,33,35}



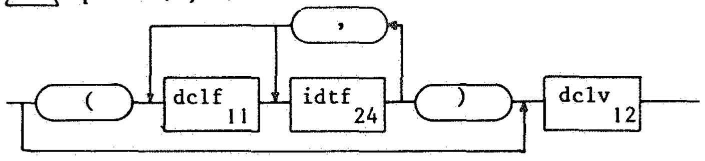
6 loop {5}



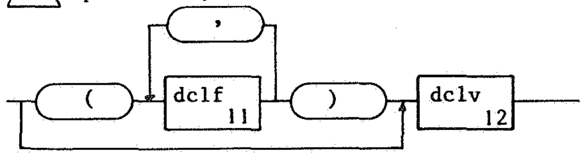
7 rtxt {5,22,23}



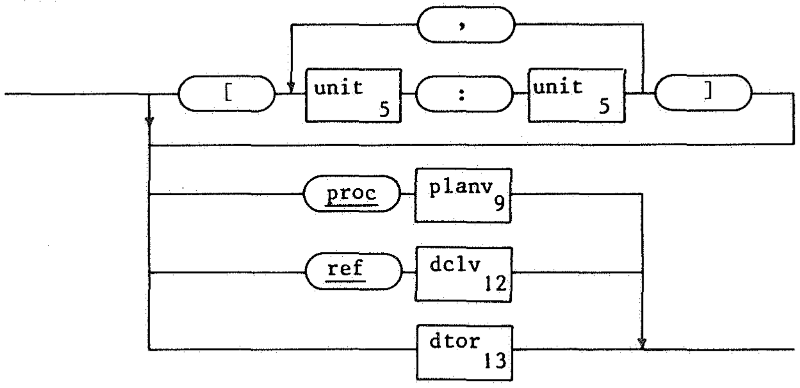
8 planf {7,20}



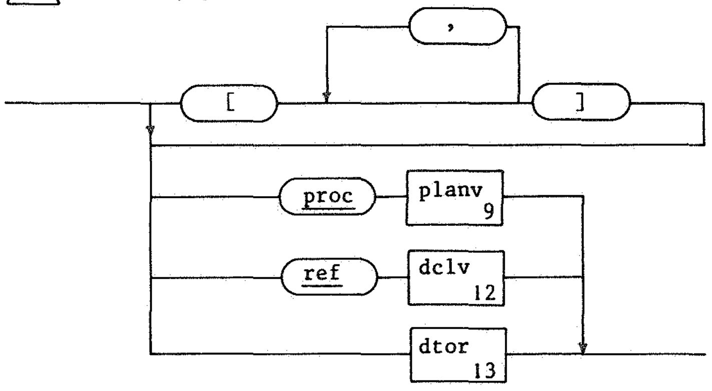
9 planv {10,11}



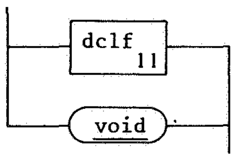
10 dcla {17,21}



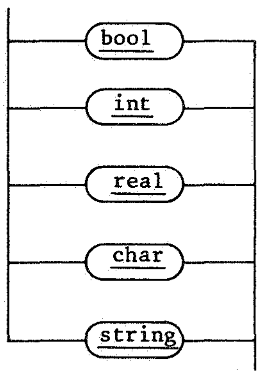
11 dclf {8,9,12}



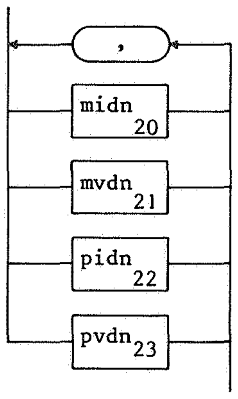
12 dclv {8,9,10,11}



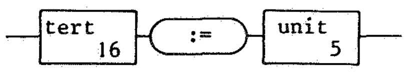
13 dtor {10,11}



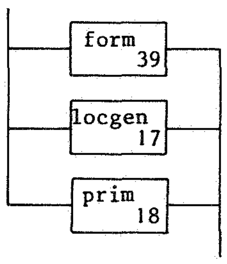
14 decln {4}



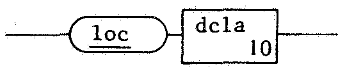
15 ass {5}



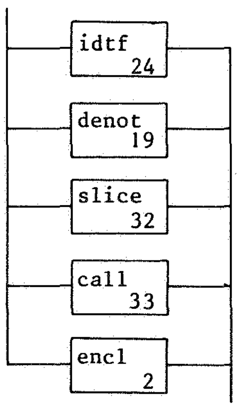
16 tert {5,15}



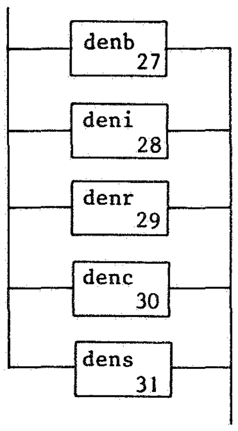
17 locgen {16}



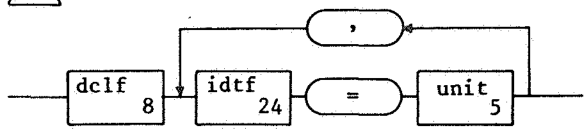
18 prim {16,32,33,38,46}



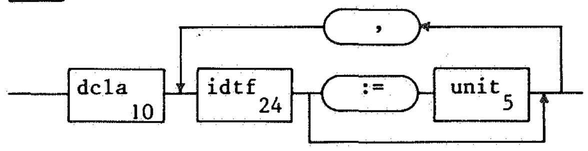
19 denot {18}



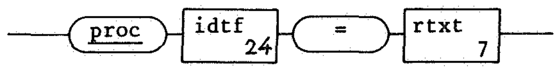
20 midn {14}



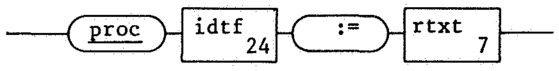
21 mvdn {14}



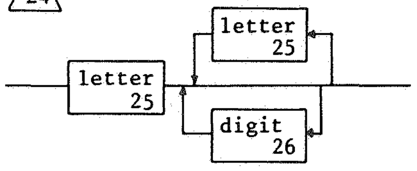
22 pidn {14}



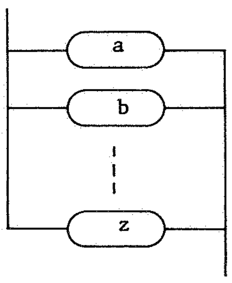
23 pvdn {14}



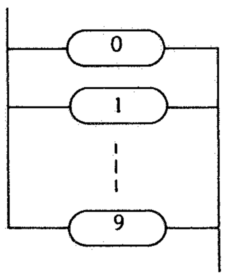
24 idtf {6,8,18,20,21,22,23}



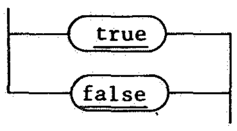
25 letter {24}



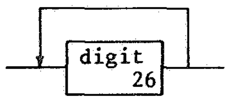
26 digit {24,28}



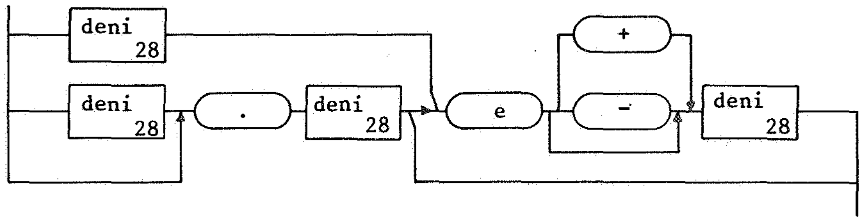
27 denb {19}



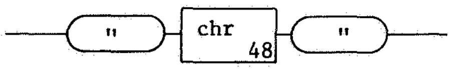
28 deni {19,29}



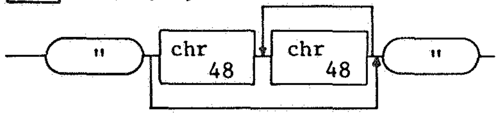
29 denr {19}



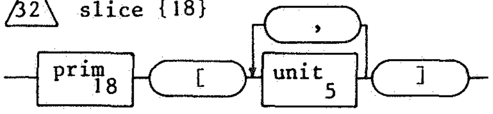
30 denc {19}



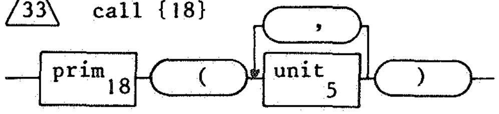
31 dens {19}



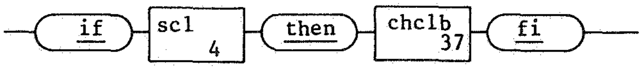
32 slice {18}



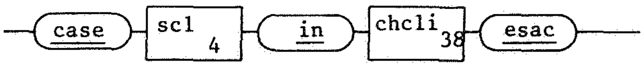
33 call {18}



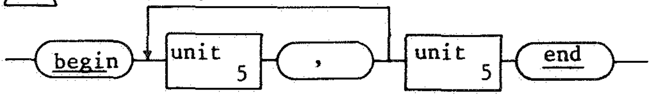
34 condcl {2}



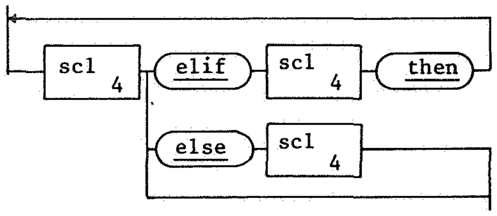
35 casecl {2}



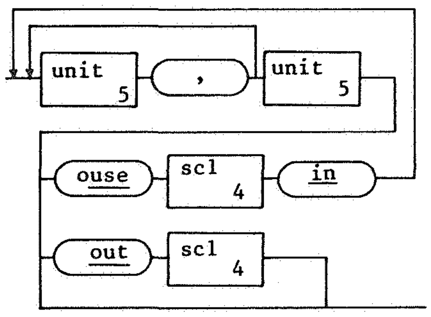
36 collcl {2}



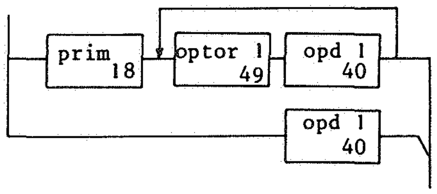
37 chclb {34}



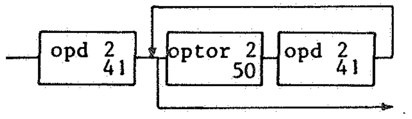
38 chcli {35}



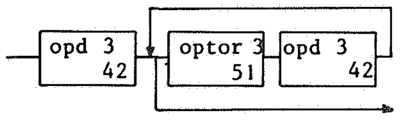
39 form {16}



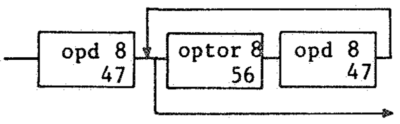
40 opd1 {39}



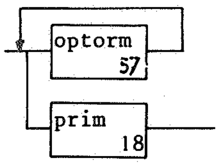
41 opd2 {40}



46 opd7 {45}



47 opd 8 {46}



48 chr {30,31}

Any printable character (including the three following characters

1) " , written as "" to avoid ambiguity

2) blank space

3) #).

| | | | |
|---|---|---|---|
| 49 | optor 1 {39} | +:= <br> -:= <br> *:= | (ref int, int) ref int |
| | | overab | |
| | | +:= <br> -:= <br> *:= <br> /:= | (ref real, real) ref real |
| 50 | optor 2 {40} | or | (bool, bool) bool |
| 51 | optor 3 {41} | and | (bool, bool) bool |
| 52 | optor 4 {42} | = <br> ne | (bool, bool) bool <br> (int, int) bool <br> (real, real) bool <br> (char, char) bool |
| 53 | optor 5 {43} | < <br> > <br> le <br> ge | (int, int) bool <br> (int, real) bool <br> (real, int) bool <br> (real, real) bool <br> (char, char) bool |
| 54 | optor 6 {44} | + <br> - | (int, int) int <br> (int, real) real <br> (real, int) real <br> (real, real) real |
| 55 | optor 7 {45} | * <br> over | (int, int) int |
| | | * <br> / | (int, real) real <br> (real, int) real <br> (real, real) real |
| 56 | optor 8 {46} | ** | (int, int) int <br> (real, int) real |

| | | |
|---|---|---|
| | lwb<br>upb | (int,[,...,]×××) int |
| 57 optorm {47} | not | (bool) bool |
| | +<br>-<br>abs | (int) int<br>(real) real |
| | abs | (char) int |
| | repr | (int) char |
| | odd | (int) bool |
| | / | (int) bool (see transput) |
| | entier | (real) int |
| | round | (real) int |
| | lwb<br>upb | ([]×××) int |
| | upb | (string) int |

Some restrictions, not implied by the syntax, are

1. In the construction
*if scl1 then scl2 else scl3 fi*
an identifier defined by a declaration in *scl1* can not be applied in *scl2* or *scl3*.

2. In the same way, a declaration in *scl1* in the construction
--- *while scl1 do scl2 od.*
is not valid in *scl2*.

3. Collateral clauses may not be nested, so, e.g.,
[,] *int a = ((1,2),(3,4))*
is not allowed, but
[] *int a = (1,2) is.*

4. There is no "rowing" coercion.
E.g. after
[1:1] *int a;*

the assignation

*a*[1] := *5*

must not be abbreviated to

*a* := *5*.

## 3. STANDARD PROCEDURES, ETC.

The identifiers listed below can be used without declarations; their mode and meaning, if necessary, are given in the second and third column.

| | | |
|---|---|---|
| pi | real | $\pi$ |
| sqrt(x) | | |
| exp(x) | | |
| ln(x) | | |
| cos(x) | proc (real) real | |
| arccos(x) | | |
| sin(x) | | |
| arcsin(x) | | |
| tan(x) | | |
| arctan(x) | | |
| random | proc real | pseudo random number generated from a uniform distribution on the interval [0,1) |
| clock | proc real | elapsed time in seconds |
| stop | proc void | termination of execution |

Transput (an explanation follows below)

| | | |
|---|---|---|
| inint(ch) | proc (int) int | |
| inreal(ch) | proc (int) real | |
| inchar(ch) | proc (int) char | |
| outint(ch,w,x) | proc (int, int, int) void | $1 \le w \le 14$ |
| outreal(ch,w,d,x) | proc (int, int, int, real) void | $\begin{cases} w \ge 1 \\ 0 \le d \le w - 1 \end{cases}$ |

| | |
|---|---|
| outchar(ch,x) | **proc** (**int**, **char**) **void** |
| outtext(ch,x) | **proc** (**int**, **string**) **void** |
| newline(ch) | **proc** (**int**) **void** |
| newpage(ch) | **proc** (**int**) **void** |
| print(x) | **proc** (**real**) **void** |
| standin | **int** |
| standout | **int** |

For input the channel numbers 1,2 or 3 may be substituted for *ch*. The channel numbers for output are 4, 5 and 6.

It is intended that *ch=1* is used for the standard file INPUT and that *ch=6* is used for the standard file OUTPUT. The identifier *standout* has the value 6, so one may write, e.g.

*newline (standout); print (i+j),*

which is more in accordance with ALGOL 68 than

*newline (6); outint (6,10,i+j).*

Similarly, the value of *standin* is 1. For *outint* and *outreal*, parameter *w* is the width, being the number of positions to be used on the external device to write the value of *x*. If $|x|$ is not too large, *outreal* will result in a fixed point representation. If it is too large, a floating point representation of *x* in *w* positions is given if $w \geq 8$. Otherwise *outreal* will write a sequence of *w* asterisks (*). The same applies to *outint*. Parameter *d* of *outreal* is the number of digits to be written to the right of the decimal point in a fixed point representation. Negative values are written by *outint* and *outreal* with a minus sign immediately preceding the first significant digit. For non-negative values a plus sign is omitted.

A special monadic operator (/) can be used to inquire if subsequent input is still possible. An input channel number (1, 2 or 3) is the operand for this operator. So the following program copies a file from channel 3 to channel 4:

```
          begin while /3
                do outchar (4, inchar (3))
                od
          end
```

The link between channel numbers and local file names is given in the next chapter.

## 4. CONTROL CARDS, EXAMPLES AND DIAGNOSTICS

The control cards (or, more generally, the commands) that can be used to compile and execute a Mini ALGOL 68 program are illustrated by means of two examples.

EXAMPLE 1: Ringcode [1)]

In this program the integer *maxc* is read, and for each integer *c* *(c=0,1,...,maxc)* the number of ones occuring in the binary representation of *c* is determined, together with the number of consecutive zeroes to the right of each one. The reverse algorithm is then carried out, yielding the original integer *c*. The contents of a complete card deck including control cards and the input value *10* for *maxc*, followed by the output of the program is given after the following more readable form of the same program.

```
begin # ringcode #
     proc code = ([] int sequence) int:
        (int code := 0;
         for k to upb sequence
         do code *:= 2 +:= 1 *:= 2 ↑ sequence [k] od;
         code),
     proc length = (int code) int:
        (int length := 0, c := code;
         while c > 0
```

1) This program, in a slightly different form, is due to Professor A. van Wijngaarden.

```
          do (odd c | length +:= 1); c ÷:= 2 od;
          length),
     proc sequence = (int code) [] int:
         (int l := length (code), c := code;
          [1:l] int sequence;
          for k to l do sequence [k] := 0 od;
          while c > 0
          do (odd c | l -:= 1 | sequence [l] +:= 1);
              c ÷:= 2
          od;
          sequence);
     int maxc = inint(1);
     for c from 0 to maxc
     do print (c); print (length (c)); newline (standout);
          [] int seq = sequence (c);
          for i to upb seq do print (seq [i]) od;
          newline (standout); print (code (sequence (c)));
          newline (standout); newline (standout)
     od
```

```
JOB,CM60000,T15.
ACCOUNT,"accountnr".
ATTACH,ALGOLM,ID=ALGOLM.
ATTACH,RUNSYS,ID=ALGOLM.
ALGOLM.
REWIND,OBJ.
COMPASS,I=OBJ,L=0,S=0.  ⎫
LOAD,RUNSYS.            ⎬       under INTERCOM to be replaced by
LGO.                    ⎭       XEQ,LOAD=LGO,RUNSYS.
7-8-9 (end of record card)
```

```
'BEGIN' #RINGCODE#
     'PROC' CODE = ([]'INT' SEQUENCE) 'INT':
         ('INT' CODE:=0;
          'FOR' K 'TO' 'UPB' SEQUENCE
          'DO' CODE *:= 2 +:= 1 *:= 2 ** SEQUENCE [K] 'OD';
          CODE),
     'PROC' LENGTH = ('INT' CODE) 'INT':
         ('INT' LENGTH := 0, C := CODE;
          'WHILE' C > 0
          'DO' ('ODD' C ! LENGTH +:= 1); C 'OVERAB' 2 'OD';
          LENGTH),
     'PROC' SEQUENCE = ('INT' CODE) [] 'INT':
         ('INT' L := LENGTH (CODE), C := CODE;
          [1:L] 'INT' SEQUENCE;
          'FOR' K 'TO' L 'DO' SEQUENCE [K] := 0 'OD';
          'WHILE' C > 0
          'DO ('ODD' C ! L -:= 1! SEQUENCE [L] +:= 1); C 'OVERAB' 2 'OD';
          SEQUENCE):
     'INT' MAXC = ININT (1);
     'FOR' C 'FROM' 0 'TO' MAXC
     'DO' PRINT (C); PRINT (LENGTH (C));
          NEWLINE (STANDOUT);
          [] 'INT' SEQ = SEQUENCE (C);
          'FOR' I 'TO' 'UPB' SEQ 'DO' PRINT (SEQ [I])'OD';
          NEWLINE (STANDOUT);
          PRINT (CODE (SEQUNECE (C))); NEWLINE (STANDOUT);
          NEWLINE (STANDOUT)
     'OD'
   'END'

7-8-9                  (end of record card)
10
6-7-8-9                (end of file card)
```

```
0     0

0

1     1
0
1

2     1
1
2

3     2
0     0
3

4     1
2
4

5     2
1     0
5

6     2
0     1
6

7     3
0     0     0
7

8     1
3
8

9     2
2     0
9

10    2
1     1
10
```

EXAMPLE 2: file copying.

By this rather trivial program the link between channel numbers and local file names is illustrated. Suppose that we want to copy the local file *INPT* to the local file *OUTPT*, by means of the program at the end of chapter 3. This can be done by writing

```
LGO,,,INPT,OUTPT.
```

instead of the simple command *LGO.* in example 1. Note that the channel number (3 for *INPT* and 4 for *OUTPT*) is simply the number of commas preceding the local file name in the *LGO*-command. In fact, the given card is an abbreviation of

```
LGO,INPUT,,INPT,OUTPT,,OUTPUT.
```

Similarly the command *ALGOLM.* in example 1 stands for

```
ALGOLM,INPUT,OBJ,OUTPUT.
```

Here other local file names may be substituted for *INPUT*, *OBJ* and *OUTPUT*.

## DIAGNOSTICS

When an error is detected during compilation or execution, an error message is issued and the compilation or execution is terminated. The error message consists of at most ten characters and may therefore be somewhat cryptical. The position in the program where the error occurred is also given by a number. At the beginning of each line in the source listing the number of positions preceding this line is printed. By this numbering the position of the error in the program can easily be determined.

## REFERENCES

1. WIJNGAARDEN, A. VAN (ed.), B.J. MAILLOUX, J.E.L. PECK & C.H.A. KOSTER, *Report on the Algorithmic Language ALGOL 68*, Numer. Math. <u>14</u> (1969) 79-218.

2. WIJNGAARDEN, A. VAN, et al. (eds.), *Revised Report on the Algorithmic Language ALGOL 68*, Springer-Verlag, Berlin-Heidelberg-New York, 1976, or: Mathematisch Centrum, Amsterdam, 1976.

3. LINDSEY, C.H. & S.G. VAN DER MEULEN, *Informal Introduction to ALGOL 68*, North-Holland Publishing Company, Amsterdam, 1976.

4. AMMERAAL, L., *Syllabus Cursus ALGOL 68, Herziene uitgave (in Dutch)*, Mathematical Centre IC 1/76, Amsterdam, 1976.

5. AMMERAAL, L., *An Implementation of an ALGOL 68 Sublanguage*, International Computing Symposium 1975, North-Holland Publishing Company, Amsterdam (1975), 49-53.

6. AMMERAAL, L., *On the Design of Programming Languages including Mini ALGOL 68*, GI-5. Jahrestagung, Lecture Notes in Computer Science Vol. 34 (1975), 500-504.