THE ATLAS AUTOCODE
MINI-MANUAL

W. F. Lunnion
G. Riding
July 1965
References & Notes


'Programming Manual for Atlas Basic Language' at I.C.T. Ltd, list no. CS348A. (Also CS348 with amendments.)

There follows a summary of the basic facilities of Atlas Autocode, as realised in COMPILER AB. [ ] are used for grouping and parenthesis, := and : - mean 'stands for' and 'the following', and a prime indicates 'any no. of, including none'; in this context these symbols are external to the language.

\( x := \) name of a real scalar
\( i := \) name of an integer scalar
\( a := \) name of a [real] array
\( y(l) := \) name of a [real] array element
\( k := \) name of an integer array
\( k(l) := \) name of an integer array element
\( v := \) name of any variable, e.g. \( x, i, a(l), k(l) \)
\( A := \) general name

**STRING** := any string of characters excluding newline & semicolon

\( E := \) real expression
\( I := \) integer expression, a special case of \( E \)
\( C := \) any constant
\( N := \) any integer, a special case of \( C \)

**STAT** := statement (q.v.) - a permissible line of program

**BP** := bound pair := \( I : I \)

\( \alpha \)-list := \( \alpha, \alpha, \ldots, \alpha \) where \( \alpha \) stands for any of the above

Characters, Names & Delimiters

Letters & digits: - A ... Z a ... z 0 ... 9
+ - * / ( ) = < > _ , . ; \& [ ] \( \alpha \beta \pi \)

Compound characters: - \( \# \# \leq \geq \% \% \Rightarrow \Rightarrow \)

All except the last of these include backspace. See also Captions.

\( L := \) letter \( D := \) digit \( P := \) prime \( A := \) name := LL'D'P'
delimiter := - + - * / ( ) , ; > < \geq \leq \Rightarrow \Rightarrow \ |

cycle repeat real integer array name if then unless
caption comment fn spec end end of program return
result and or begin stop routine switch

Labels & Jumps

**LABEL** := simple or switch label := - \( N : \) A(N):
**JUMP** := - \( \Rightarrow N \) \( \Rightarrow A(I) \)

Switches must be declared, together with their ranges, (see DECL) and the name A must be local, DECLs and routine headings should not be labelled; blank lines may be.
OPERND := operand: -  v C (E) |E| function

OPERTR := operator: -  +  -  /  \n
E := real expression: E' OPND [OPTR OPND]'
I := integer expression: - E, but with all OPNDS of type integer,
    and such that the result is an integer

C := constant: fixed or floating point, using a to imply
    a base of 10 for the exponent, e.g. fifteen may be
    punched: 15 0015.00 .15\alpha 2 1500\alpha 2

Operator precedence: ( ) are evaluated first, then \',
    then * and /, then + and -. Otherwise the
    operations are performed in order of occurrence,
    e.g. a + b - (d - c \# e \# f)
    means (a + (b - (d - (c \# e) \# f)))

ASST := assignment: -  v = I  x = E  a(I) = E

Statements

\emptyset := \not\exists \geq \leq \geq

SC := simple condition: - E \notin E  (GC)

GC := general condition: - SC \{and SC\}'  SC \{or SC\}'

UI := unconditional instruction: - ASST routine call jump
    return result = E  stop caption STRING

CI := conditional instruction: - if GC then UI
    unless GC then UI  UI if GC  UI unless GC

DECL := declaration: - real x-list integer i-list
    array [a-list(BP-list)] - list  OR  real array ...
    integer array [k-list(BP-list)] - list
    switch [A-list(i;i)] - list  routine spec [See Routines]

ST := unlabelled statement: - UI CI DECL routine heading
    begin end  end of program  cycle  repeat | STRING
    [see also Monitoring]

STAT := statement := LABEL' ST

Blocks

A block is a list of statements enclosed between begin and end.
Blocks are 'open': they may be entered and left only via begin and
end resp. Blocks may be nested to any depth. A label in a block or at
its end may only be referred to by an instruction in the block and
not in a sub-block. The 'scope' of a name is the block at the START
of which it is declared (the block to which it is 'local'), together
with any blocks which may be nested within (to which it is 'non-local').
A name may be used within its scope, and nowhere else; but non-local
names may be re-declared.

A program is a block with end replaced by end of program and
followed by optional data, (together with the necessary supervisor
material).
Routines

RT := routine type :- routine real fn integer fn
Corresponding exit :- return result = E result = I
R := name of a routine of the appropriate type
Routine spec :- RT spec R(FP-list)
Routine heading :- RT R(FP-list)
Routine call :- R(AP-list)

FP := formal parameter :-
    integer name i
    real name x
    array name a
    integer array name k
    integer i
    real x
    routine R
    real fn R
    integer fn R
    R

AP := actual parameter :-
    i, k(i)
    v
    a
    k
    I
    E
    R

A routine is a named block with parameters; the routine heading
replaces begin, and the corresponding exit is inserted. The FPs in
the heading have the force of declarations inside the routine; but
when the FP is of routine type, a spec must be inserted (from which
the RT may be omitted).

A routine is 'closed': it can only be entered via a call, which
causes it to be obeyed with the FPs in its heading replaced by the
APs in the call. An integer or real FP will be assigned the value
of the AP at time of call ('call by value'). A ...name FP will be
assigned the actual store location of the AP, so that e.g. assignments
to the FP alter the AP; but should the AP be an array element, its
subscripts will be treated as integer parameters and remain fixed
throughout the routine ('call by simple name')

In a list of FPs of the same type, the type delimiters after the
first may be omitted. In the case of a parameterless routine, the
(FP-list) and (AP-list) are omitted.

Cycles

cycle 1 = I1, I2, I3
    [list of statements]
    repeat
Here I1, I2, I3 are all of the form I, and such that (I3 - I1)/I2
is a positive integer or zero. I1, I2, I3 are evaluated at the
start of the cycle, and remain unaltered throughout it.
Cycles can be nested to any depth.
Standard Functions

real fn
sin(E)  cos(E)  tan(E)  log(E)  exp(E)  sqrt(E)  arctan(E1, E2)
 [= arctan(E2/E1), in (-π/2, π/2) if E1 > 0, in (π/2, 3π/2) if E1 < 0]
arcsec(E) [in (-1, 1)]  arccos(E) [in (0, π)]
radius(E,E)  fracpt(E)  mod(E)  |E|

integer fn
intpt(E) [intpt(-3.73) = -4, intpt(3.73) = 3]  int(E)  parity(I)

Standard functions may not be substituted for FPs of routine type
in routine calls (see Routines).

Input and output

routine
select input (I)  I refers to stream in JOB descr.
select symbol (I)  See table of numerical equivalents
read symbol (I)  
print symbol (I)  
print (E, I1, I2)  Punches I1+I2+2 characs., or I1+1
  if I2 = 0, unless I1 is too small
print f1 (E, I)  Punches I+7 characs,
read (v-list)  Reads nos. from data 'tape' in order,
punched as constants except for optional sign, terminated
by space or newline, e.g.  +15.0  -1.501  .1501
Initial spaces and newlines are ignored,
read binary (I)  5, 7, or 12 least significant bits
punch binary (I)  

  tab  Character positions for tab:
  0 8 16 24 32 48 64 80 96 112 128 144 159
Tab advances the character position at least two spaces.
newline , newlines (I)
spaces , spaces (I)
runout (I)  no effect on lineprinter
newpage  thirty newlines on seven-hole punch

integer fn
next symbol  See table of numerical equivalents.

Does not advance data 'tape'

Captions & quotes

Instead of the numerical equivalent, the character itself
may be used enclosed by quotes. An entire string of characters can
be output by the statement

caption STRING
In captions or between quotes use the symbols
& or ½ to denote space
& or ½ or " space underlined
; or ½ or " semicolon
& or & or " newline

e.g. print symbol (97); print symbol (65)
     print symbol ('a'); print symbol ('$')

     caption as

all have the same effect.

**Numerical equivalents**

<table>
<thead>
<tr>
<th>Character</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>33</td>
</tr>
<tr>
<td>z</td>
<td>58</td>
</tr>
<tr>
<td>a</td>
<td>97</td>
</tr>
<tr>
<td>z</td>
<td>122</td>
</tr>
<tr>
<td>o</td>
<td>16</td>
</tr>
<tr>
<td>9</td>
<td>25</td>
</tr>
</tbody>
</table>

newline 4  +  29  ß  91  $  1296266
( 8  -  30  ½  92  $  14807
) 9  .  31  #  11164  $  1895382
, 10  '  32  ≠  3599  $  14167
π 11  space  65  ≤  11034  $  1435530

The numerical equivalent of a compound character is \((128^x + y)\) \(128^y + z\),
where \((x, y, z)\) are the equivalents of its constituents and \((x > y) > z\).

**Notes on punching**

Tab is converted to multiple space on input.

If the printout (including erases) looks right, the tape is right.

In the program:

A statement is terminated by newline or semicolon.

at the end of a line causes the newline to be ignored, i.e. the
statement continues on to the next line and the line
number is not advanced.

Spaces, underlined spaces, erased characters and superfluous
terminators are ignored.

and ² are converted to .§ and ½ on input.

Comments may be inserted by means of

    comment STRING     or     | STRING

# is an alternative to ≠.

In the data:

Erased characters are ignored.

For the supervisor material (JOB descr., etc.) see elsewhere.
Monitoring

monitor statement: compile queries ignore queries

compile array bound check stop array bound check
fault [N-list -> LABEL]-list ASST ?

Array subscripts are tested dynamically for overflow if they appear
in the program between compile ... and stop array bound check, and ?
- which causes the value in the preceding ASST to be printed each
time the ASST is obeyed - is similarly controlled by compile ... and
ignore queries.

Faults detected at compile time are noted in the program 'map'
printed during compiling, and the program is not then entered. Dynamic
faults, unless trapped, cause the private monitor to print the current
routine and line number and a summary of the stack (working space),
then terminate the run. Faults occurring just before a JUMP, return,
etc. may escape detection till after the jump has been obeyed.

Some faults may be trapped by the statement fault ..., causing a
jump to the LABEL (which must be simple) if fault N subsequently
occurs. The stack is then restored to its extent at the time when
the fault statement was obeyed; but some variables may have been
altered in the meantime. fault ... is dynamic but not nested - the
LABEL used is that in the last fault ... obeyed containing N - so it
should normally be confined to the outermost block.

Common trappable faults: -

<table>
<thead>
<tr>
<th>TRAP NO.</th>
<th>NATURE OF FAULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Division by 0</td>
</tr>
<tr>
<td>2</td>
<td>Exponent overflow</td>
</tr>
<tr>
<td>4</td>
<td>More store required</td>
</tr>
<tr>
<td>5</td>
<td>Square root of no. &lt; 0</td>
</tr>
<tr>
<td>6</td>
<td>Logarithm of no. &lt; 0</td>
</tr>
<tr>
<td>8</td>
<td>Trig. fn. out of range</td>
</tr>
<tr>
<td>9</td>
<td>No more data</td>
</tr>
<tr>
<td>14</td>
<td>Data fault: spurious character at start of no.</td>
</tr>
<tr>
<td></td>
<td>(this character is the 'next symbol')</td>
</tr>
<tr>
<td>16</td>
<td>': real no. instead of integer</td>
</tr>
<tr>
<td></td>
<td>(the no. is read in off the 'tape')</td>
</tr>
</tbody>
</table>

Sample program

The program overleaf is badly written (superfluous instructions
and cumbersome method) to involve more facilities of the language,
and its blocks are delineated for emphasis. The output is shown after.
begin

comment tabulate the binomial coefficients \( \binom{i}{j} \), with \( i \) down the left margin and \( j \) across the bottom, for \( i = 0 \) (1) \( n \)

real fn spec fact (integer p)
routine spec binom (real name ans, integer p, q)
integer n ; read (n)
caption ABINOMIAL\$COEFFICIENTS$A
-> 1 unless n < 0
caption Ns<spn ; -> 2
1: begin

array B (0:n, 0:n)
integer i, j ; real x
cycle i = 0, 1, n
    newline
    print (i, 2, 0)
    cycle j = 0, 1, i
        binom (x, i, j)
        B(i, j) = x
        print (B(i, j), 6, 0)
    repeat
    newline ; spaces (3)
cycle j = 0, 1, n
    print (j, 6, 0)
    repeat
stop
end

routine binom (real name ans, integer p, q)

comment puts p\(q\) in ans
ans = \( \frac{\text{fact}(p)}{\text{fact}(q) \times \text{fact}(p-q)} \)
end

real fn fact (integer p)

comment result = p !
result = 1 if p = 0
if p \neq 0 then result = p \times \text{fact}(p-1)
end

2: end of program
BEGIN BLOCK NO = 91 ADDRESS =00115050
9 BEGIN BLOCK NO = 94 ADDRESS =00116000
27 END BLOCK OCCUPIES 192 LOCATIONS
28 BEGIN ROUTINE <BINOM> NO = 93 ADDRESS =00121010
31 END ROUTINE <BINOM> OCCUPIES 52 LOCATIONS
32 BEGIN REAL FN <FACT> NO = 92 ADDRESS =00121660
36 END REAL FN <FACT> OCCUPIES 43 LOCATIONS
37 END BLOCK OCCUPIES 350 LOCATIONS

PROGRAM ENTERED

BINOMIAL COEFFICIENTS

0  1
1  1  1
2  1  2  1
3  1  3  3  1
4  1  4  6  4  1
5  1  5  10 10 5  1
0  1  2  3  4  5

logging information (supervisor)